

# Mathematik

Vektor  $\vec{x} = (x_1, x_2, \dots, x_n)$

Multiplikation  $\alpha * \vec{x} = (\alpha * x_1, \alpha * x_2, \dots)$

Addition  $\vec{x} + \vec{r} = (x_1 + r_1, x_2 + r_2, \dots)$

Linearkombination  $\vec{\sigma} = (\alpha * \vec{p}) + (\beta * \vec{q}) + (\gamma * \vec{r})$

Länge:  $\vec{p} = (x, y, z) : |\vec{p}| = \sqrt{x^2 + y^2 + z^2}$

Skalarprodukt:  $\vec{x} * \vec{r} = \sum_{i=0}^{n-1} x_i * r_i$

Winkel  $\vec{a} * \vec{b} = |\vec{a}| * |\vec{b}| * \cos(\phi)$  mit  $\cos(\phi) = \frac{\vec{a} * \vec{b}}{|\vec{a}| * |\vec{b}|}$

Vektorprodukt (Kreuzprodukt)  $\vec{a} \times \vec{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix}$

Ebenen:  $\vec{p} = \vec{q} + \alpha * \vec{r} + \beta * \vec{s}$

Dreieck  $\vec{A} + \alpha * (\vec{B} - \vec{A}) + \beta * (\vec{C} - \vec{A})$

## 2D Transformation

**Translation** um den Vektor  $\vec{t}$

**Skalierung** Stauchung oder Streckung

**Spiegelung** • an der x-Achse  $S = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$   
(Vorzeichenumkehr für y-Koordinaten)

• an der y-Achse  $S = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$  (Vorzeichenumkehr für x-Koordinaten)

• am Ursprung  $S = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$  (entspricht Spiegelung an x- und y-Achse)

**Scherung**  $S = \begin{pmatrix} 1 & S_x \\ S_y & 1 \end{pmatrix}$

**Rotation mit Polarkoordinaten**  $P' = (r, \phi + \theta)$ ;

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix}$$

**Koordinatentransformation**

$$P' = T * P = \begin{pmatrix} x_x & x_y \\ y_x & y_y \end{pmatrix} * \begin{pmatrix} P_x \\ P_y \end{pmatrix}$$

Viele zweidimensionale Transformationen lassen sich als 2x2 Matrizen darstellen (Ausnahme Translation).

**Homogene Vektorräume** Vektorraum mit zusätzlicher Dimension w, als 3-Tupel  $(x, y, w)$ . Entspricht dem kartesischen Vektor  $(\frac{x}{w}, \frac{y}{w})$ . oft  $w = 1$  gewählt (1="Punkt", 0="Richtung"; kartesische Zahlenebene).

Skalierung, Projektion, Spiegelung:

$$\begin{pmatrix} F_x & 0 & 0 \\ 0 & F_y & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} F_x * x \\ F_y * y \\ 1 \end{pmatrix}$$

Für Skalierung gilt  $F_x, F_y > 0$ , für uniforme Skalierung gilt zusätzlich  $F_x = F_y$ .

Mit  $F_x = 0$  oder  $F_y = 0$  kann eine Projektion auf die y-, bzw x-Achse ausgedrückt werden.

$F_x = -1$  oder  $F_y = -1$  entspricht einer Spiegelung an der y- bzw x-Achse. Sind beide Faktoren  $F_x = F_y = -1$ , wird damit eine Spiegelung am Ursprung ausgedrückt.

$$\text{Scherung: } \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} x + a * y \\ y \\ w \end{pmatrix}$$

$$\text{Rotation: } R_\theta * P = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x & y & 1 \end{pmatrix} =$$

$$\begin{pmatrix} x \cos(\theta) - y \sin(\theta) \\ x \sin(\theta) + y \cos(\theta) \\ 1 \end{pmatrix}$$

**2D Transformation mit Fixpunkten** Operation aus bekannten Operationen zusammengesetzt

- verschiebe von P in den Ursprung (Translation  $T_{-P}$ )
- rotiere um Ursprung ( $R_{\theta_0}$ )
- Translation zurück ( $T_{+P}$ )
- In Matrixform sieht die Operation wie folgt aus:  
 $T_{+P} * R_{\theta_0} * T_{-P} * \text{Objekt}$

**Invertierung**

- Transformation  $T_{\Delta x, \Delta y}^{-1} = T_{-\Delta x, -\Delta y}$
- Skalierung  $S_{F_x, F_y}^{-1} = S_{\frac{1}{F_x}, \frac{1}{F_y}} = \begin{pmatrix} \frac{1}{F_x} & 0 & 0 \\ 0 & \frac{1}{F_y} & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- Rotation  $R_{-\theta} = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} = R_\theta^T$
- Verknüpfungen  $(A * B * C)^{-1} = C^{-1} * B^{-1} * A^{-1}$

**Affine Abbildung**

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ 1 \end{pmatrix}$$

- die letzte Zeile der affinen Matrix bleibt immer 0,0,1
- paralleles bleibt bei affinen Abbildungen stets parallel

**Homogene Transformation in 3D**

werden durch eine 4-dimensionale homogene Vektorraum repräsentiert

**Ebenen** sinnvoll: (a,b,c,d) wobei (a,b,c)=(nx,ny,nz) und d der Abstand der Ebene zum Ursprung

- Ebene definiert durch 3 Punkte

$$\begin{pmatrix} x_1 & x_2 & x_3 & 0 \\ y_1 & y_2 & y_3 & 0 \\ z_1 & z_2 & z_3 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

- Translation um Vektor  $(\Delta x, \Delta y, \Delta z)$

$$\begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Skalierung um Faktor  $F_x, F_y, F_z$

$$\begin{pmatrix} F_x & 0 & 0 & 0 \\ 0 & F_y & 0 & 0 \\ 0 & 0 & F_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotation um z-Achse

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotation um die x-Achse

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotation um die y-Achse

$$\begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Kameratransformation** Kamera ist definiert durch

- Lage des Augpunktes E (in Weltkoordinaten)
- Blickrichtung D
- Oben-Vektor U ("view up vector", senkrecht zu D)

**Projektion**

**Orthogonale Projektion**

- Projektionsebene ist parallel zur XY Ebene
- Projektionsrichtung stets parallel zur z-Achse (rechtwinklig zur Projektionsebene)
- z Koordinaten werden auf gleichen Wert gesetzt

**Schiefwinklige Parallelprojektion**

- typische Parallelprojektion mit 2 Parametern
- Projektionsebene ist parallel zur XY Ebene
- Projektionsrichtung hat zwei Freiheitsgrade und ist typischerweise nicht orthogonal zur Projektionsebene
- Projektionsrichtung (Schiefe) ist über 2 Winkel parametrisierbar

$$\text{Herleitung } P = \begin{pmatrix} 1 & 0 & -\cos(\alpha) * f & 0 \\ 0 & 1 & -\sin(\alpha) * f & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- es gilt:  $x' = x - \cos(\alpha) * f * z$  und  $y' = y - \sin(\alpha) * f * z$

## Zentralperspektive

- entspricht einer Lochkamera bzw etwa dem einäugigenSSehen
- Augpunkt im Ursprung des Kamerakoordinatensystems
- Projektionsfläche ist eine Ebene parallel zu XY Ebene
- Eigenschaften
  - perspektivische Verkürzung
  - parallele Linien des Objekts fluchten oft in einen Fluchtpunkt

$$\begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} d * x \\ d * y \\ 1 \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \frac{d*x}{z} \\ \frac{d*y}{z} \\ \frac{1}{z} \end{pmatrix}$$

## Fluchtpunkte

- hat ein Modell parallele Kanten oder parallele Striche in Texturen, dann ergibt sich für jede solche Richtung r in der Abbildung ein Fluchtpunkt, auf den diese parallelen Kanten/Striche hinzu zu laufen scheinen
- es gibt jedoch Ausnahmen, bei denen Paralleles in der Abbildung Parallel bleibt (z.B. horizontale Kanten der Schwellen in der Abbildung)
- Da es beliebig viele Richtungen geben kann, sind auch beliebig viele Fluchtpunkte in einem Bild möglich
- Rotationen können Fluchtpunkte ändern, Translationen jedoch nichtbrechender
- Ermittlung: aus Richtung r und Augpunkt eine Gerade, dann schneidet diese Gerade die Projektionsfläche im Fluchtpunkt für die Richtung r.

## Modellierung

### Geometrische Modellierung

computergestützte Beschreibung der Form geometrischer Objekte

### Boundary Representation (B-Rep)

- Beschreibung durch die festlegung begrenzender Oberflächen
- Darstellungsform eines Flächen- oder Volumenmodells
- beschreibt Objekt durch begrenzende Oberflächen
- sind schnell verarbeitbar
- Definition eines Objekts erfolgt über einen vef-Graph (vertex, edge, face)
  - Knotenliste: beinhaltet Koordinatenpunkt
  - Kantenliste: für jede Kante werden zwei Punkte referenziert
  - Flächenliste: für jede Fläche wird Reihenfolge von Kanten angegeben

## Szenengraph

- Szene: dreidimensionale Beschreibung von Objekten, Lichtquellen und Materialeigenschaften mit virtuellen Betrachter
- Szenegraph: hierarchische Gruppierung der Objekte in einer Szene

## Rendering

Render-Pipeline: Geometrisches Objekt- $\rightarrow$  Transformieren- $\rightarrow$  Vertex Shader- $\rightarrow$  Raster Konvertierung- $\rightarrow$  Fragment Shader- $\rightarrow$  Ausgabebild

## Vertex Shader

- verarbeitet alle Eckpunkte (Vertices) mit Shader
- ermöglicht eine Beeinflussung der Objektform
- Transformation der 3D Position auf 2D Koordinaten für Bildschirm
- Input
  - Vertices relevanter Objekte der Szene
  - gewünschte Transformation
- Output
  - auf Bildschirm projizierte 2D Koordinaten
  - zugehörige Tiefeninformationen

## Model View Projection

- Gegeben
  - Modell als Vertices mit kartesischen 3D koordinaten und definierten Dreiecken
  - Kamera (3D Position, Ausrichtung) welche das Modell betrachtet
- Umsetzung
  - $M = T * R * S$  zuerst Transformation von Modellraum (gegebenen Koordinaten) in Weltkoordinaten (Model)
  - $V = T_V^{-1} * R_V^{-1}$  danach Transformation in Kameraruum, für einfachere Projektion (View)
  - abschließende projektion auf Kamerabildebene und Umrechnung in Bildraum (Projektion)
- Ergebnis
  - Model-View-Projektion-Matrix  $P * V * M = MVP_{Matrix}$
  - Anwendung der MVP ergibt Bildraumprojektion des Modells  $p'_m = P * V * M * p_m$
  - MVP-Matrix nur einmal berechnet

## Effiziente geometrische Algorithmen und Datenstrukturen

**Bintree** - effizientes Suchen und Einfügen in eindimensionale Domänen - logarithmische Komplexität pro Zugriff möglich - Gefahr: lineare Komplexität, wenn nicht balanciert

analog zu Quad- und Octrees: - typisch ost Teilung in Mitte (bisektion) - Bereiche mit homogenem Inhalt (gleiche Farbe/ keine Elemente) werden nicht weiter unterteilt - Komprimierungseffekt

- eine (meist quadratische) Fläche kann bei Bedarf in vier gleichgroße Quadranten unterteilt werden - Bedarf entsteht, wenn die Fläche keine homogenen Eigenschaften aufweist (z.B. bei unterschiedlich gefärbten Pixeln). D.h. Flächen (bzw Quadranten) werden solange unterteilt, bis sie homogen sind - Anwendung - Geometrische 2D Objekte können in hierarchische Struktur einsortiert werden, wodurch die räumliche Suche nach diesen Objekten beschleunigt wird - Effiziente Speicherung von Rasterbildern wird möglich (Komprimierung, da nur strukturierte Bereiche unterteilt werden)

Jeder Knoten hat 0 oder 8 Kindknoten. Damit wird bei Bedarf ein 3D Bereich in 8 Unterbereiche unterteilt. Geometrische Objekte (z.B. 3D Punkte) können in diese hierarchische Strukturen einsortiert werden, wodurch die räumliche Suche nach diesen Punkten beschleunigt wird.

Beispiel Punktsuche:

Suche einen Punkt mit Koordinaten (x,y,z) im Octree. Rekursive Suche von der Wurzel. In jedem Schritt wird einer von 8 möglichen Pfaden im Teilbaum ausgewählt - $\rightarrow$  Zeitaufwand Tiefe des Baumes  $O(\log n)$

- mehrdimensionaler binärer Baum (k-dimensional) - unterteilt z.B. abwechselnd in x-,y-, und z-Richtung (deshalb binärer Baum) - Teilung nicht zwangsläufig mittig (wie bei Octre) - $\rightarrow$  an Daten angepasst - jeder neue Punkt teilt den Bereich in dem er einsortiert wird; pro Hierarchiestufe stets wechsel der Teilungsrichtung - ein Octree lässt sich auf einen kd-Baum abbilden, beide Baumarten haben daher vergleichbare Eigenschaften

KD-Baum mit der Median-Cut Strategie:

Der Median-Cut teilt Daten in zwei gleich großen Hälften. Damit wird der Baum garantiert balanciert und die tiefe wird minimal. Damit wird das  $O(\log n)$  Verhalten garantiert. Probleme können bei lokalen Häufungen (Cluster) auftreten. Die Mediantcut Strategie bewirkt eine degenerierte globale Teilung des Gesamttraumes aufgrund von lokalen Situationen. Dies wirkt sich ungünstig bei der Suche weit weg vom Cluster aus - vollständig balanciert - unnötige Unterteilung weit weg vom Cluster (Artefakt)

Vergleich Middlecut-Strategie: - nicht balanciert - keine Unterteilung weit weg vom Cluster

Praxis: Kompromiss Strategie, Mischung zwischen Median und Mitte. Bei Situations-Analyse: Explizite Abkapselung des Clusters. Außerdem: In (2D, 3D) kann man natürlich das strikt zyklische Abwechseln der Achsen aufgeben und bei fast

ebenen/linearen Strukturen lokal nur noch in zwei bzw 1 Richtung(en) unterteilen!

Verallgemeinerung des kd-Baums. Trennebenen sind nicht nur achsenparallel. Unterteilung in beliebigen Richtungen, adaptiv an Modellflächen angepasst.

Beachte: Trennebenen die an einer Objektebene anliegen können dennoch weiter wegliegende Objekte schneiden. BSP-Tree führt bei konvexen Polyedern zu entarteten Bäumen sehr einfache Abfrage (nur ein Vergleich  $j$  in jeder Koordinatenrichtung, wie bei kd-Baum) einfach zu erstellen (min, max), dafür nicht optimale Packungsdichte bei schräger Lage der Objekte.

passen sich besser der räumlichen Ausrichtungen an, lassen sich auch leicht transformieren (Rotation bei Animation). Jedoch schwieriger zu erstellen (Wahl der Richtung), komplexere Überlappungsberechnung (Transformation, Ebenengleichung). D.h. OBB-trees werden typischerweise weniger tief, weniger räumliche Abfragen dafür wesentlich mehr Berechnungsaufwand pro Rekursionsstufe. Polyeder mit festen vorgegebenen Richtungen (z.B. 45 Grad). Eigenschaften zwischen AABB und OBB. Bessere Raumausnutzung als AABB, weniger Transformationen als OBB.

Schnelle 3D Überlappungstest (Abstand der Mittelpunkte  $j$  Summe der Radien). Langgezogene Objekte können mit mehreren Hüllkugeln (Bounding Spheres) begrenzt werden um besseren Füllgrad zu erreichen. BS sind bis auf die Lage der Kugelmittelpunkte invariant gegenüber Rotation (eignen sich für Kollisionserkennung bewegter Objekte/ Echtzeit-Computer-Animation).

- Kollisionserkennung in Computeranimation (Computerspiele). Reduktion der potenziellen Kollisionspaare durch räumliche Trennung - Beschleunigung des Echtzeitrenderings großer Datenmengen. Reduktion des Aufwands durch Culling (Weglassen) Raytracing/Strahlverfolgung, Ray picking mit KD-Baum: Vorverarbeitung, abspeicherung von Objekten (Dreiecken) im kd-Baum  $O(n \log n)$

Strahl/Objektschnitt: (als rekursive Suche im kd-Baum)  $treeIntersect(p,d)$ : Findet Schnittpunkt des Strahls (Punkt  $p$ , Richtung  $d$ ) mit den im Baum gespeicherten Dreiecken und liefert die Beschreibung des nächsten Schnittpunktes bzw  $t$ =unendlich, falls kein Schnittpunkt existiert.

$triangleIntersect(node,p,d)$ : Findet Schnittpunkt des Strahles (Punkt  $p$ , Richtung  $d$ ) mit einer Menge von Dreiecken in  $node$   $subdivide(node, p, d, tmin, tmax)$ : Findet rekursiv den nächstgelegenen Schnittpunkt (kleinstes  $t$ ) des Strahls ( $p,d$ ) mit den Dreiecken in oder unterhalb von  $node$  im Parameterbereich  $tmin \dots tmax$

Komplexität Strahl Objektschnitt (Extremfälle + typischer Fall) 1. Extremfall (beinahe) konvexes Objekt (max 2 Schnitte möglich) - hat ca die Komplexität einer räumlichen Punktsuche, also dem Aufwand zur Untersuchung einer Baumzelle (finden + dortige Dreiecke testen)  $O(\log n)$  2. Extremfall "Polygonnebel" (viele sehr kleine Dreiecke im Such-Volumen) - Annahme: alle Zellen enthalten konstante kleine Anzahl von Dreiecken -  $j$  Aufwand proportional zur

Anzahl durchlaufener Baumzellen - Anzahl dieser Zellen ist proportional zur Länge des Strahls durchs Volumen, da der 1. Schnitt sehr wahrscheinlich mitten im Volumen oder gar nicht stattfindet -  $j$  Anzahl ist proportional zur Seitenlänge des Suchvolumens - bei  $n$  Dreiecken im Suchvolumen ist die Anzahl  $t$  der zu untersuchenden Zellen also ca  $t = O(\sqrt{n})$  -  $j$  Suchaufwand pro Strahl folglich  $O(\sqrt{n} \log(n))$  typische Szene: Suchaufwand fast immer wesentlich besser als  $O(n)$ . Aufwand oft zwischen beiden extremen, bei mittlerer Dichte sogar eher  $O(\log n)$ .

Effektiver Zeitaufwand für Raytracing (RT): - absoluter Gesamtaufwand zum Raytracing einer Szene (z.B. in ms) ist (auch) proportional zur Anzahl der Strahlen - Annahme: 1 Strahl pro Pixel (keine Rekursion), typische Bildgröße sei 1 Mio Pixel, Szene haben mittlere Komplexität (1 Mio Polygone) - Performancebeispiel: - Stand 2006, PC mit 1 CPU ca 1 Mio Strahlen/Sek -  $j$  1 fps - Stand 2019, PC mit "RTX 2080 Sup23" ca 10 Giga Strahlen/Sek -  $j$  10000 fps - rekursives RT (Reflexion, Brechung, Schattenstrahlen etc) entsprechend mehr Strahlen, d.h. weniger Performance - Parallelisierung einfach möglich (z.B. da Pixel voneinander unabhängig berechenbar) -  $j$  früher CPU-basiert, heute eher GPU - 2019 mit entsprechender Hardware: rekursives Echtzeit Raytracing möglich

Surface Area Heuristic (SAH): - Annahme: Strahl  $i$ , trifft Zelle  $j$  mit Wahrscheinlichkeit  $P(i,j)$ , zudem sei  $n_j$  die Anzahl Dreiecke in Zelle  $j$ , - Aufwand für Raytracing pro Zelle proportional zur Baumtiefe ( $O(\log n)$ ) für balancierte Bäume, wird nicht weiter betrachtet) sowie die Anzahl der dortigen Dreiecke  $n_j$ ; beachte  $n_j$  wird hier nicht als konstant angenommen -  $j$  Gesamtaufwand für Strahl  $i$  sei also  $\sum(P(i,j) * n_j)$

Heuristik: große Zellen mit wenigen Dreiecken, senken Gesamtaufwand - Schätzung:  $P(i,j)$  ist etwa proportional zur Oberfläche einer Zelle (auf großer Oberfläche treffen mehr Strahlen auf) - die SAH optimiert auf jeder Teilstufe im Baum das Produkt der Zellgröße mal Anzahl Dreiecke im Teilbaum. Für den kd-Baum gilt bei der Unterteilung des Bereichs  $D$  in Richtung  $k$ :  $D_k = D_{k_{links}} + D_{k_{rechts}}$

Bei ungleicher Verteilung der Dreiecke (z.B. Cluster) enthalten dann große Zellen wenige oder keine Dreiecke und Baum ist nicht balanciert -  $j$  implizite Abtrennung des Clusters vom Rest des Baums (vgl Middle-Cut-Strategie)

Problematik: Abspeicherung ausgedehnter Objekte Punkte haben keine Ausdehnung und können an einem eindeutigen Ort im kd-Baum abgelegt sein. Ausgedehnte Objekte (Kreise, Kugeln, Rechtecke, Dreiecke, Hüllquader, etc) können räumlich mehrere Blatt-Zellen überlappen. Ein solches Objekt müsste dann in mehreren Blattzellen einsortiert sein.

1. Ansatz: Auftrennung von Objekten, d.h. Objekte müssen an der Zellgrenze aufgeteilt werden. Einsortierung der Teilobjekte in passende Zellen. Geht gut für Dreiecke 2. Ansatz: Keine Unterscheidung zwischen Blattknoten und inneren Knoten. In diesem Ansatz werden Objekte soweit oben im Baum einsortiert, dass sie keine Zellgrenze schneiden. Nachteil: auch relativ kleine Objekte müssen in große Zellen einsortiert werden, wenn sie deren Unterteilungsgrenze schneiden 3.

Ansatz: Loose Octree (überlappende Zellen): die Zellen des Octrees werden so vergrößert, dass sie mit ihren direkten Nachbarn in jeder Richtung um 50

Algorithmus zum Zeichnen einer Strecke:

Endpunktkoordinaten sind nach Projektion in die Bildebene passend auf die Fensterkoordinaten skaliert und auf ganzzahlige Werte (Pixelkoordinaten) gerundet.

- von J. Bresenham 1965 bei IBM entwickelt - Grundidee: Effizient durch Verwendung von Ganzzahlen, Vermeiden von Multiplikation/Division sowie Nutzung einer inkrementellen Arbeitsweise

Die Linie geht zwischen den Endpunkten nicht durch ganzzahlige Gitterpunkte. Da nur ganzzahlige Pixel-Koordinaten gesetzt werden können müssten auch zwischenpunkte zuerst genau berechnet werden und dann auf ganzzahlige Pixelwerte gerundet werden. Dies ist unzuverlässig und ineffizient. Zur Herleitung des effizienten Bresenham-Algorithmus führen wir den Mittelpunkt  $M$  als Referenzpunkt ein. Ferner seinen der jeweils aktuellen Punkt  $P$ , der rechts von im liegende  $E$  (east) und der rechts oben liegende  $NE$  (north-east) benannt. - die Linie wird als Funktion  $y = f(x)$  repräsentiert:  $y = \frac{\delta y}{\delta x} * x + B$  - in implizierter Form:  $d : F(x,y) = \delta y * x - \delta x * y + B * \delta x = 0$  - für Punkte auf der Linie wird  $F(x,y) = 0$  - für Punkte unterhalb der Linie wird  $F(x,y) > 0$  - für Punkte oberhalb der Linie wird  $F(x,y) < 0$  Herleitung mit Einschränkung:

Steigung der Linie  $m$  ( $1 < m < 1$ ), Mittelpunkt  $M =$  Punkt vertikal zwischen zwei möglichen Pixeln  $E$  und  $NE$ . Ausgehend von bereits gesetzten Pixel  $P$  auf der Linie für den nächsten Mittelpunkt  $M$ . Für gefundenen Mittelpunkt, berechne die Distanzfunktion  $d$ . Daraus Kriterium zur Wahl des nächsten Pixels: Falls  $F(x_p + 1, y_p + \frac{1}{2}) > 0$  wird das nächste Pixel  $NE$ , andernfalls  $E$ .

Insgesamt acht verschiedene Fälle: 1. Oktant( $\delta y < \delta x$ ) 2. Oktant( $\delta y > \delta x$ ) 3. Oktant( $\frac{\delta y}{\delta x} < 0$ ) 4. Oktant( $\frac{\delta y}{\delta x} < -1$ ) 5. -8. Oktant( $\delta x < 0$ )

Effekte der Rasterisierung: Aliasing - Treffenstufeneffekt bei gerasterten Linien - Regelmäßigkeiten werden verstärkt vom Auge wahrgenommen

Das Auflösungsvermögen des Auges für Punkte sei  $e$ . Strukturen wie Linien (bestehend aus vielen Punkten) werden durch Mittelwertbildung (Fitting) vom Auge viel genauer als lokalisiert. Eine Stufe wird umso eher erkannt, je länger die angrenzenden Segmente sind.

Grundidee des Anti-Aliasing 1. Original der Linie 2. Statt der Linie wird ein Rechteck mit der Breite von einem Pixel betrachtet 3. Graustufen der darunter liegenden Pixelflächen entsprechen dem jeweiligen Überdeckungsgrad Praktische Umsetzung mit vereinfachtem/effizienterem Algorithmus 1. Rasterkonvertierung der Linie bei doppelter örtlicher Auflösung (Supersampling) 2. Replizieren der Linie (vertikal und/oder horizontal) um Linienbreite näherungsweise zu erhalten 3. Bestimmung des Überdeckungsgrades pro Pixel in der ursprünglichen Auflösung (Downsampling) 4. Bestimmung des Farbwertes entsprechend des Überdeckungsgrades

Problem: - Ausgabe von Linien/Polygonen auf Rastergeräten muss auf vorgegebenem Raster erfolgen - Farbvariation ist zwar möglich, Farbberechnung muss aber effizient erfolgen Ohne Antialiasing: - es erfolgt ein einfacher Test über die Pixelkoordinate - verwendet Farbe in der Pixelmitte Ideales Antialiasing: Hat wegen der beliebig komplexen Geometrie allgemein einen sehr/zuhohen Aufwand! Ansatz für eine "reale Lösung eine ideale Berechnung von Farbwerten nach dem Überdeckungsgrad ist allgemein beliebig aufwendig und daher praktisch irrelevant - Gesucht werden Ansätze mit gut abschätzbarem/konstanten Aufwand - "reales Antialiasing beruht in der Regel auf der Verwendung von mehreren Samples pro Pixel, d.h. Berechnung dieser n Samples statt nur einem (typisch: n-facher Aufwand) Beim Supersampling-Ansatz wird die Graphik zunächst in höherer Auflösung gerendert (z.B. 4-fach), und dann aus den Samples ein Farbwert gemittelt (z.B. per arithmetischem Mittel). Wie wirksam ist das Antialiasing bei 4x-Supersampling - Ohne Anti-Aliasing kommt pro Pixel genau eine Sampleposition zum Zuge. Das Pixel wird demnach gefärbt oder nicht gefärbt: Das sind zwei mögliche Stufen. - Bei vier Subpixeln können minimal 0 und maximal 4 Subpixel im (Makro-)Pixel gesetzt sein, d.h. es sind Intensitäten von 0-1 Der Zusammenhang ist einfach: Es gibt immer eine Abstufung mehr als Subpixel pro Pixel. - Beim idealen Antialiasing entsprechend Flächenbedeckungsgrad gibt es "beliebig" viele Abstufungen (aus Aufwandsgründen aber praktisch nicht relevant). - Gibt es eine Formabhängigkeit? Ja, z.B. bei 45° gibt es z.B. nur eine Zwischenstufe, und zwar je nach Phasenlage mit 25 Minderung der Formabhängigkeit → Supersampling mit Rotated Grids - Kleine Winkel zu den Achsen führen zu langen SStufen" der digitalen Polygonkante - Bessere Verhältnisse der Graustufen ergeben sich für flache Winkel, wenn statt des ordered grid ein "rotated grid" für das Supersampling verwendet wird. - Rotated grids sind dafür bei anderen Winkeln etwas schlechter als das ordered grid. Dies wird aber kaum wahrgenommen, da dort die Treppen der digitalen Geraden kürzer sind. Vorteile: - Gute Graustufen bei sehr flachen Kanten zur Zeilen- oder Spaltenrichtung. - Optimaler Winkel liegt bei ca.  $20 \circ$  bis  $30 \circ$  (z.B.  $\arctan(0.5)$   $26,6 \circ$ ). - Sehr dünne Linien (dünner als 1 Pixel) bleiben auch bei Bewegung zusammenhängend bzw. sichtbar (Vermeidung von "Line Popping"). Realisierung der Subpixelberechnung durch einen Superbuffer (Supersampling in entsprechend großem Buffer) oder mehrere Multisamplebuffer (Multisampling in mehrere Buffer der ursprünglichen Größe): 1. Superbuffer - Nachteil (bei rotated grid): Anpassung der Rasterkonvertierung an verschobene Positionen erforderlich - Vorteil: Verwendung von mehr Texturinformation (Textur wird subpixelgerecht eingetragen) 2. Multisamplebuffer - Mehrfachrendering in normaler Größe mit leicht versetzter Geometrie (Vertexverschiebung pro Sub-Bild) - Vorteil: keine Veränderung im Rendering - Nachteil: nur ein Texturwert pro Makro-/Sub-Pixel

Gezielter Ressourceneinsatz: 1. Kantenglättung: - Effizienzsteigerung durch Beschränkung auf reine Kantenglättung möglich! - Kanten vs. Flächen: Anzahl der Kantensamples oft wesentlich kleiner als Anzahl der Flächensamples - Aliasing bei Kanten/Mustern in Texturen schon beim Auslesen der Werte aus der Pixeltextur unterdrückbar - Kantensamples bekannt als: separate Linien oder Berandung von Polygonen/Dreiecken 2. adaptives Sampling: - statt feste Anzahl von Samples kann die Anzahl nach dem Bedarf gesteuert werden Quincunx → Überfilterung Ausgangspunkt ist das 2x Multisampling mit rotated grid! Entsprechend dem Grundprinzip des Anti-Aliasing sollen Aliasingartefakte durch Erzeugung höher aufgelöster Information reduziert werden. Bei dem vorausgesetzten Muster ist der Informationszuwachs durch die doppelte Anzahl von Samples gekennzeichnet, egal wie ausgewertet wird! Quincunx-Verfahren: - Information für die Kantenglättung beruht nach wie vor auf 2 Subpixeln - Entspricht einer zusätzlichen Tiefpass-Überfilterung. Durch die Unschärfe sehen Polygonkanten glatter aus. - Harte Kanten sind gar nicht mehr möglich, dadurch wird auch Szappeln an Polygonrändern reduziert. - Aber Nachteil: Texturinformation, die nur zu 2 Subpixeln gehört, wird verschmiert! Kombinationen und Pseudozufälliges Supersampling: - Grundsätzlich ist die Kombination von Supersampling, Multisampling und Quincunx möglich und einige Graphikkarten unterstützen solche Überlegungen in jeweils festgelegten Grenzen (ordered oder rotated grid, gemischtes Sampling, Downsampling-Ansatz). Der Gewinn hält sich bisher in Grenzen, bei unqualifiziertem Mix ergeben sich eher Nachteile. - Bei Überwindung der für Füllrate und Bandbreite gegebenen Grenzen überwiegen die Vorteile des Supersamplings. - Ordered grid und rotated grid weisen bei bestimmten Strukturklassen Vor- und Nachteile auf. Die verbleibenden Artefakte wiederholen sich bei großen Flächen, so dass derartige Muster vom Menschen oft als störend empfunden werden. → aus diesen und ähnlichen Überlegungen → Ansätze für die Weiterentwicklung: - pseudozufällige Auswahl von Abtastmustern für das Supersampling - nachträgliche Abminderung regelmäßiger Strukturen durch vorsichtiges Verrauschen (Rauschfilter) - entfernungsabhängiges Antialiasing pseudozufällig: - Samples können nur an n vordefinierten Positionen stattfinden (Sample-Positionsmuster). - Je nach Methode werden daraus m Positionen für das Sampling zufällig ausgewählt (beachte:  $m \leq n$ ) - Anzahl der Muster als kombinatorisches Problem:  $m$  aus  $n$  (ohne Wiederholungen) Beim Anti-Aliasing zur Glättung von Polygonkanten kommt für das Downsampling die Mittelwertbildung in Frage (lineare Filterung (2x - AA), bilineare Filterung (4x - AA)), gleichgültig ob ordered oder rotated grid. Beim pseudozufälligen Supersampling ist entsprechend der "frei gewählten" Positionen der Subpixel zu modifizieren (z.B. Gewichte nach Abstand der Abfragepositionen zur Makropixelposition).

Ansatz: - finde die Pixel innerhalb des Polygons - weise ihnen Farbe zu - dabei zeilenweises Vorgehen, pro Rasterlinie: - für jede Polygonkante: - schneide die Polygonkante mit der aktuellen Bildzeile (→  $x_s$ ) - füge Schnittpunkt  $x_s$  in eine Liste ein - sortiere Schnittpunkte der aktuellen Bildzeile in x-Richtung - Paritätsregel: fülle die Pixel jeweils zwischen ungeradem und nächstem geradem Schnittpunkt (Pixel zwischen geraden und ungeraden Schnittpunkten aber nicht!) Beachte: Die Schnittpunkte in floating point zu berechnen und zu runden ist ineffizient. Wir suchen, ähnlich wie beim Bresenham-Algorithmus, einen inkrementellen Ansatz mit Ganzzahl-Arithmetik. Allgemeinerer Sicht auf die Füll- bzw. Auswahlstrategie: Ein Pixel wird mit der Farbe des Polygons gefüllt, das sich rechts von ihm befindet. Sollte dort eine Kante sein, so wird die Farbe des oberen Polygons verwendet. Grundsätzlich könnten beliebige Richtungen als Referenzrichtung zur Farbbestimmung gewählt werden. Dann müssten die zuvor besprochenen Regeln oder der gesamte Algorithmus entsprechend angepasst werden. Effiziente Ermittlung der Schnittpunkte von Polygonkante und Rasterzeile: - Vorbetrachtungen: - Polygonkanten werden stets von unten nach oben bearbeitet - horizontale Polygonkanten müssen nicht bearbeitet werden (geschieht in Scanline) → im Algorithmus stets  $m \neq 0$  -  $d_y = y_1 - y_0$  ist stets positiv (auch nie 0) -  $d_x = x_1 - x_0$  kann positiv und negativ sein - damit können 4 Bereiche unterschieden werden - Idee - Berechnung von  $x$  bzw  $y$ : 
$$y = y_0 + m(x - x_0), y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0), x = x_0 + \frac{1}{m}(y - y_0), x = x_0 + \frac{x_1 - x_0}{y_1 - y_0}(y - y_0)$$
 - Zwar sind die  $x$ - bzw.  $y$ -Werte immer noch nicht ganzzahlig, jedoch können sie als rationale Zahlen explizit mit Zähler und Nenner repräsentiert werden. - Die Rundung (nächstes  $x$  oder  $y$  erreicht?) kann inkrementell ermittelt werden. - Die Rundungsregel für Bruchwerte hängt davon ab, ob es eine linke oder rechte Kante ist. Links wird z.B. aufgerundet (Pixel ist auf oder rechts v. der Kante). Edge-Tabelle: - Verkettete Liste (oder Array, siehe unten) für die nicht horizontalen Kanten. - Sortierung nach der Scan-Line, wo die Kanten beginnen (unteres Ende,  $y_0$ ). - Innerhalb der Scan-Line wiederum Liste (nach  $x_0$ -Werten sortiert). Je nach Implementierung werden z.B.  $x_0, y_1$ , sowie Zähler und Nenner gespeichert. Active-Edge-Tabelle: - Die AET speichert alle Kanten, die die gegenwärtige Scan-Line schneiden. - Die Liste hat die gleiche Struktur wie eine Zeile der ET. - Die Kanten werden gelöscht, wenn das obere Ende der Kante erreicht ist. Bemerkung zu Scan Convert Polygon: 1. Es existiert immer eine gerade Anzahl Kanten. Bei konvexen Polygonen sind immer null oder zwei Kanten in der AET. Die Sortierung ist dadurch trivial bzw. entfällt bei konvexen Polygonen. Bei vielen Grafikbibliotheken (z.B. OpenGL) beschränkt man sich auf konvexe Polygone. Nichtkonvexe Polygone müssen daher vorher in konvexe Komponenten zerlegt werden. Dafür ist das Füllen dieser Polygone danach effizienter. 2. Dieser Teil entspricht einem Schleifendurchlauf der Prozedur EdgeScan. Die Unterscheidung zwischen linker und rechter Kante wird beim Auffüllen der Pixel gemacht.



monochromatischen Lichtquelle  $x(\lambda_0)$  in diesem RGB Farbraum lässt sich wie folgt berechnen:  
 $x(\lambda_0) = p * r + \gamma * g + \beta * b$  - Achtung: die Lösung hängt von den Wellenlängen der verwendeten Grundfarben r,g,b (Primärvalenzen) ab. Nimmt man andere Grundfarben, ergeben sich andere Werte für  $p, \gamma, \beta$  - RGB ist nicht gleich RGB! Man muss eigentlich immer die Wellenlängen der verwendeten Grundfarben r,g und b mit angeben!

Innere Farbmischung:  $F = p * r + \gamma * g + \beta * b$

Äußere Farbmischung:

die gemischte Farbe Cyan wird zwar als derselbe Buntton wie die Referenzfarbe F wahrgenommen, jedoch weniger gesättigt. Um die beiden Farben gleich aussehen zu lassen wird der Referenzfarbe F etwas Rot beige mischt. Damit sind beide Farben gleich ungesättigt. Das Verfahren wird äußere Farbmischung genannt:  $F = p * r + \gamma * g - \beta * b$ . Um die aus Blau und Grün gemischte Farbe Cyan voll gesättigt aussehend zu lassen, müsste Rot aus der Mischfarbe subtrahiert werden. Dies ist allerdings technisch nicht realisierbar. Durch die negative Farbvalenz wird das Modell jedoch theoretisch konsistent und es lassen sich alle Farben durch Mischen von Rot, Grün und Blau darstellen. Daraus wird ein vollständiges RGB-Farbmodell abgeleitet.

Idee: - es werden drei linear-unabhängige Größen benötigt - zur Beschreibung der Farbempfindung - zur (technischen) Reproduktion der Farbempfindung - zunächst werden folgende Werte gewertet - die additive Mischung als Reproduktionsmethode - drei Primärfarben Rot, Grün, Blau - drei linear unabhängige Größen spannen stets einen 3D Raum auf - die RGB Werte werden den drei orthogonalen Achsen dieses Raumes zugeordnet

Darstellung des RGB Farbraums: - alle mit drei Farblichtquellen technisch (additiv) erzeugbaren Farben liegen innerhalb eines Würfels - Im Koordinatensystem befindet sich Schwarz, diagonal gegenüber weiß. - auf der Raumdiagonalen liegen dazwischen die Graustufen

Bei entsprechender Normierung liegen die vom RGB Farbsynthesystem erzeugbaren Farben im Einheitswürfel. Zunächst wird der Begriff Intensität eingeführt:  $I = \frac{R+G+B}{3}$ . Der Ausschnitt aus der Ebene konstanter Intensität, der im Einheitswürfel liegt, wird im Interesse der einfachen Darstellung als Farbebene (Farbtafel) genutzt. Dabei bleibt die Ordnung der Farbvalenzen erhalten. Die Länge —F— der Farbvalenz bzw die Intensität geht verloren. Die in der Ebene konstanter Intensität liegenden Werte definieren die Chrominanz durch welche die Farbwertanteile erfasst werden (zwei reichen aus da 2D). Es kann auch die Projektion der Ebene auf RG (grau überlagert) als Farbtafel genutzt werden, ohne die Ordnung der Farborte zu stören. Vorteil: orthogonales rg-System

RGB Farbtafel:

Alle Farben gleicher Buntheit (unterscheiden sich nur in der Länge von F) führen zum gleichen Farbort, der durch die Farbwertanteile r,g,b beschrieben wird:

$$r = \frac{R}{R+G+B}, g = \frac{G}{R+G+B}, b = \frac{B}{R+G+B} \Leftrightarrow r+g+b = 1$$

Aus dem rechten Teil der Gleichung folgt mit  $b = 1 - r - g$ , dass sich die Buntheit allein durch r und g darstellen lässt (entspricht  $R^2$ ). Die Farbwertanteile lassen sich bei bekanntem Farbort in der Farbtafel nach der angegebenen Konstruktionsvorschrift ermitteln oder direkt ablesen.

Um eine Relation zwischen der menschlichen Farbwahrnehmung und den physikalischen Ursachen des Farbreizes herzustellen, wurde das CIE-Normvalenzsystem von der Internationalen Beleuchtungskommission (CIE) definiert. Es stellt die Gesamtheit der wahrnehmbaren Farben dar.

Farbkörperunterschiede:

Es finden sich Unterschiede welche Farbbereiche nach dem CIE Normalvalenzsystem von den jeweiligen Systemen dargestellt werden können: - menschliche Farbwahrnehmung ca 2-6 Mio Farben - Monitor: ca 1/3 davon. Bei Monitoren wird die additive Farbmischung verwendet, da die einzelnen Lichtquellen aufsummiert werden. - Druckprozess: meist deutlich weniger Farben. Bei Druckern werden einzelne Farbschichten auf Papier gedruckt und das resultierende Bild wird über die subtraktive Farbmischung bestimmt.

Subtraktive Farbmischung:

Je nachdem welche Farbe ein Material hat, werden entsprechende Farbanteile absorbiert oder reflektiert. Eine gelbe Oberfläche sieht gelb aus, da sie das Blau aus weißem Licht absorbiert, aber Rot und Grün reflektiert.

Achtung: Dies gilt nur für die Bestrahlung mit weißem Licht. Wird beispielsweise ein gelbes Blatt mit blauem Licht bestrahlt, dann wirkt es schwarz, da das blaue Licht vom gelben Blatt absorbiert wird.

Grundfrage: Was ist Licht? - Teil der elektromagnetischen Strahlung - ist für das menschliche Auge wahrnehmbar - Lichtspektrum liegen zwischen 380 nm und 780 nm - Farbe entspricht der Wellenlänge - längere Wellenlängen = weniger Photonenenergie - durch Überlagerungen vieler Frequenzen erscheint das Licht weiß

Radiometrie: - Wissenschaft von der Messung elektromagnetischer Strahlung - Größen sind physikalische Einheiten (ohne Berücksichtigung des menschl. Sehens)

Photometrie: - Messverfahren im Wellenlängenbereich des sichtbaren Lichtes (Messung mithilfe eines Photometers) - lassen sich aus den radiometrischen Größen, bei bekanntem Spektrum bestimmen - berücksichtigen die wellenlängenabhängige Empfindlichkeit des Auges

Photon: - Elementarteilchen der elektromagnetischen Wechselwirkung - besitzen keine Masse - Energie und Impuls sind proportional zur Frequenz - kürzere Wellenlänge = höhere Frequenz = höhere Energie

Strahlungsenergie (radiant energy): - durch Strahlung (elektromagnetische Wellen) übertragene Energie - entspricht dem Produkt von Photonenanzahl und der Energie der Photonen - Formelzeichen: Q - Einheit: J (Joule) - photometrisches Äquivalent: Lichtmenge (luminous energy)

Strahlungsleistung (auch Strahlungsfluss, engl. radiant flux, radiant power): - transportierte Strahlungsenergie in einer bestimmten Zeit - Formelzeichen:  $\phi$  - Einheit: W (Watt) -

Berechnung:  $\phi = \frac{Q}{t}$  - photometrisches Äquivalent: Lichtstrom (luminous flux, luminous power)

Zusammenhang zwischen Radiometrie und Photometrie: In der Radiometrie wird sich mit objektiven Messgrößen beschäftigt, in der Photometrie gibt es jeweils eine entsprechende Messgrößen, bei denen die spektrale Empfindlichkeit des menschlichen Auges mit einfließt. - Beispiel: - radiometrisch: Strahlungsleistung  $\phi_e$  gemessen in Watt W - photometrisch: Lichtstrom  $\phi_v$  gemessen in Lumen lm - Verknüpfung von Radiometrie und Photometrie erfolgt über das photometrische Strahlungsäquivalent:  $K = \frac{\phi_v}{\phi_e}$  - gibt die Empfindlichkeit des menschlichen Auges an - radiometrische Größe: Index  $e$  für energetisch - photometrische Größe: Index  $v$  für visuell Die radiometrischen Größen gewichtet mit dem photometrischen Strahlungsäquivalent K sind somit die photometrischen Größen.

Ausbreitung eines Strahls: - geradlinig von einer Quelle zum Ziel, - Richtung ändert sich durch Brechung - an Oberflächen tritt Reflexion und Streuung auf - eine Strahlungsquelle sendet dabei Strahlen in alle Raumrichtungen unter einem gewissen Raumwinkel aus

Der Steradian ist eine Maßeinheit für den Raumwinkel, der von der Mitte M einer Kugel mit Radius r aus gesehen eine Fläche von  $r^2$  auf der Kugeloberfläche einnimmt.

$\Omega = \frac{\text{Fläche}}{\text{Radius}^2} = \frac{A}{r^2} sr$  Eine komplette Kugeloberfläche  $A_k$  beträgt allg.  $A_k = 4\pi r^2$ , entspricht also einem Raumwinkel  $\Omega$  von  $\frac{A_k}{r^2} = 4\pi \approx 12,5 sr$ . Ein Steradian = 1sr entspricht einem Öffnungswinkel  $\alpha$  von ca.  $65,54^\circ$

- auch Intensität, engl. radiant intensity - Strahlungsleistung in eine Raumrichtung mit Raumwinkel  $\Omega$  emittiert wird - Formelzeichen: I - Berechnung:  $I = \frac{\phi}{\Omega}$  - photometrisches

Äquivalent: Lichtstärke (luminous intensity)

Beispiel: Berechnen Sie die Strahlstärke einer Lampe mit einem Öffnungswinkel von  $180^\circ$  und einer Strahlungsleistung von 20W.

$$\alpha = 180 \rightarrow A = 2\pi r^2; \phi = 20W; \Omega = \frac{A}{r^2} = 2\pi; I_e = \frac{\phi_e}{\Omega} = \frac{20}{2\pi} \approx 3,2 \frac{W}{sr}$$

Energieübertragung zwischen zwei Flächen: Eine Fläche  $A_r$  strahlt Licht auf eine Fläche  $A_i$  ab.

Frage: Wie viel Lichtleistung von einer infinitesimalen abstrahlenden Fläche  $A_r$  wird auf einer Fläche  $A_i$  empfangen? - der Abstand zwischen den beiden infinitesimalen Flächen beträgt r - die Flächen stehen nicht notwendigerweise senkrecht zur Ausbreitungsrichtung des Lichts (gerade Verbindungslinie zwischen den Flächen) - Wir projizieren daher die abstrahlende und die empfangende Fläche jeweils in Ausbreitungsrichtung. Die projizierten Flächen nennen wir  $A'_r$  und  $A'_i$ . - Wir betrachten Punktlichtquellen von der abstrahlenden Fläche  $A_r$ , welche ihre Strahlungsleistung in den Raumwinkel  $\Omega$  abgeben. -  $\Omega$  ist somit die in Abstrahlrichtung reduzierte Fläche  $A'_i$ , projiziert auf die Einheitskugel:  $\Omega = \frac{A'_i}{r^2}$  - Die übertragene Energie nimmt quadratisch zu r ab - engl. radiance - Strahlstärke welche von einer Sendefläche  $A_r$  in eine bestimmte Richtung abgegeben wird - Formelzeichen: L - photometrisches Äquivalent: Leuchtdichte (auch Luminanz,

engl. luminance) - Berechnung:

$$L = \frac{I}{A_r} = \frac{I}{\cos(\phi_r) * A_r} = \frac{\phi}{\cos(\phi_r) * A_r * \Omega} - \phi_r \text{ ist der Winkel}$$

zwischen der Normalen n und der Abstrahlrichtung (von der abstrahlenden Fläche  $A_r$  zur empfangenden  $A_i$ )

Leuchtdichte (Luminanz) als Vorstufe der Helligkeit: -

Strahlungsleistung bewertet mit der spektralen

Empfindlichkeitsfunktion des menschlichen Auges für das

Hellempfinden - Das menschliche Auge hat seine maximale

Empfindlichkeit, bei einer Wellenlänge von 555 nm (gelbgrün)

- 1 Lumen ist definiert als der Lichtstrom einer 1,464 mW

starken 555-nm-Lichtquelle mit 100

- auch Strahlungsflussdichte, engl. irradiance -

Strahlungsleistung durch die bestrahlte Fläche  $A_i$  bzw.

Strahlstärke die auf die Empfängerfläche trifft - Formelzeichen

: E - Berechnung:  $E = \frac{\Phi}{A_i}$  - photometrisches Äquivalent:

Beleuchtungsstärke (auch Lichtstromdichte, engl. illuminance)

- erweitert:  $E = \frac{\Phi}{A_i} = \frac{L * \cos(\phi_i) * \cos(\phi_r) * A_r}{r^2}$

Radiometrische (physikalische) und Photometrische (unter Berücksichtigung des menschlichen Auges) Größen

— Symbol — Radiometrie (energetisch  $e$ ) — Photometrie

(visuell  $v$ ) — — — — —  $Q$  — Strahlungsenergie

*Joule* — Lichtmenge  $lm * s$  —  $\Phi$  — Strahlungsleistung

Watt  $W$  — Lichtstrom Lumen  $lm$  —  $I$  — Strahlstärke  $\frac{w}{sr}$

— Lichtstärke Candela  $cd$  —  $E$  — Bestrahlungsstärke  $\frac{W}{m^2}$

— Beleuchtungsstärke Lux  $\frac{lm}{m^2}$  —  $L$  — Strahllichte  $\frac{w}{sr * m^2}$

— Leuchtdichte  $\frac{cd}{m^2}$  —

Nach dem Auftreffen auf einer opaken Oberfläche wird die Strahlung spektral unterschiedlich stark und geometrisch auf unterschiedliche Weise reflektiert. Es können 2 Idealfälle der Reflexion unterschieden werden: - ideal spiegelnde Reflexion (Einfallswinkel = Ausfallswinkel) - ideal diffuse Reflexion

Aus den zwei Idealfällen der reflexion werden weitere (gemischte) Fälle abgeleitet: - spekuläre Reflexion (diffus und gerichtete Reflexion) - gemischte Reflexion: ideal diffus, gerichtet diffus und ideal spiegelnd

Bei der Betrachtung der Reflexion ist offensichtlich die Art der Bestrahlung und insbesondere auch die Richtung der Einstrahlung zu beachten.

Lichtquelle im Unendlichen; Irradiance  $E = \frac{A'_i}{A_i} I_{in} = I_{in} \cos(\phi)$

Eingestrahle Strahlstärke I in durch  $A'_i$  verteilt sich durch die Projektion auf die größere Fläche  $A_i$  Die Bestrahlungsstärke E (Irradiance) ist dadurch proportional zum Vergrößerungsfaktor der Fläche abgeschwächt.

In Richtung Betrachter reflektierte Strahlstärke  $I_{out}$  Aufgrund von Interferenz phasengleicher Lichtstrahlen → Projektion auf Normalenrichtung  $\frac{I_{out}}{E_{refl}} = \cos(\phi)$  - Senkrecht zur Oberfläche:

Maximale Kohärenz (Addition) - Parallel zur Oberfläche: n Keine Kohärenz (Auslöschung)

Annahme kohärentes Licht: Parallel zur reflektierenden Oberfläche findet sich zu jeder Punktlichtquelle immer eine gleichphasige Punktlichtquelle im Abstand  $\frac{\lambda}{2}$  - Auslöschung parallel zur Fläche,

$$\frac{A_r}{A_r} = \frac{1}{\cos(\phi)} \rightarrow L = \frac{I_{out}}{\cos(\phi)} = I_{refl}$$

Ein Betrachter mit flachem Blickwinkel sieht Licht aus größerer Fläche  $A_r$  durch Kombination dieser Effekte, kürzt sich der Einfluss des Betrachterwinkels  $\cos(\phi)$  weg und es bleibt nur der Einfluss des Lichteinfallswinkels übrig: Strahllichte des reflektierten Lichtes:  $L = I_{in} * k_d(\lambda) * \cos(\phi)$  Spekuläre (gestreut spiegelnde) Reflexion: - Speckles (Flecken), bzw. (Micro-) Facetten sind einzeln jeweils ideal spiegelnd: Einfallswinkel  $\phi = \text{neg. Ausfallswinkel} = -\phi$ . - Die Ausrichtung der Microfacetten weichen von der Gesamtlächennormalen ab. → Statistische Abweichung von der Flächennormalen (z. B. Gauß-Verteilung) - dadurch Streuung des Lichts (Keule) um den Winkel  $\theta$  der idealen Spiegelung herum - Je größer der Winkel  $\theta$  zwischen idealer Spiegelrichtung und Richtung zum Betrachter, desto schwächer ist die Reflexion - Modellierung meist per  $\cos^k(\theta)$  (Phong-Beleuchtungsmodell) - nicht physikalisch begründet.

Gestreute Spiegelung im Phong Modell mit

$L = I * k_s * \cos^k(\theta)$  - glänzende Oberfläche: großer Exponent k (16,...,128); kleine Streuung  $\epsilon$  - matte Oberfläche: kleiner Exponent k (1,...,2); große Streuung  $\epsilon$

Energieerhaltung → Verhinderung der Abnahme bei großen Exponenten → Für die Energieerhaltung wird ein zusätzlicher Normierungsfaktor benötigt: - physikalisch nicht korrekt:

$L = I * k_s * \cos^k(\theta)$  - gebräuchliche Normierung  $\frac{k+2}{2\pi}$  somit:

$$L = I * k_s * \frac{k+2}{2\pi} * \cos^k(\theta)$$

- Wegen der spektralen Unterschiede bei der Reflexion bleiben wir bei den spektralen physikalischen (radiometrischen) Größen! - Erst im Auge bzw. im Bildsensor erfolgt die Wandlung in die wellenlängenintegralen photometrischen (colorimetrischen) Größe!

Zunächst ideal diffus remittierende weiße Flächen ( $\beta(\lambda) = 1$ ): - Die von den Quellen in die Fläche  $dA$  eingetragene Leistung führt zu einer Bestrahlungsstärke  $E_\lambda$  - Bei vollständiger Reflexion  $\beta(\lambda) = 1$  ist  $E_\lambda = R_\lambda$  (spektrale Radiosity, spezifische spektrale Ausstrahlung). - Der zugehörige spektrale Strahlungsfluss  $d\phi = R_\lambda * dA = E_\lambda * dA$  wird bei ideal diffusen streuenden Oberflächen gleichmäßig über den Halbraum verteilt, wobei die Strahllichte (Lambertsches Gesetz) konstant ist.

- englisch Bidirectional Reflectance Distribution Function, BRDF - eine Funktion für das Reflexionsverhalten von Oberflächen eines Materials unter beliebigen Einfallswinkeln - Ziel: Oberfläche möglichst realistisch und physikalisch korrekt darstellen - nach gewählter Genauigkeit sehr komplex - in der Computergrafik wird meist eine vereinfachte Variante gewählt um Rechenzeit zu sparen - erstmals 1965 definiert (Fred Nicodemus):  $f_r(\omega_i, \omega_r) = \frac{dL_r(\omega_r)}{dE_i(\omega_i)} = \frac{dL_r(\omega_r)}{L_i(\omega_i) \cos(\theta_i) d\omega_i}$  - Eine BRDF beschreibt wie eine gegebene Oberfläche Licht reflektiert. - Das Verhältnis von reflektierter Strahllichte (radiance)  $L_r$  in eine Richtung  $\vec{r}$  zur einfallenden Bestrahlungsstärke (irradiance)  $E_i$  aus einer Richtung  $\vec{i}$  wird "bidirectional reflectance distribution function" (BRDF)

genannt. -  $p(\lambda) = \frac{L_r}{E_i} = [\frac{1}{sr}]$  - Die BRDF (für jeden Punkt x)

ist eine 5-dimensionale skalare Funktion:  $p(\lambda, \phi_e, \theta_e, \phi_i, \theta_i)$  - Keine Energie-Einheiten, nur Verhältniszahl! - Kann durch Messung für verschiedene Materialien bestimmt werden (Messkamera/Normbeleuchtung) - Eigenschaften der BRDF: - Reziprozität: ( $\lambda$ ) ändert sich nicht, wenn Einfalls- und Ausfallsrichtung vertauscht werden (wichtig für Ray-Tracing).

- ( $\lambda$ ) kann anisotrop sein, d.h. der Anteil des reflektierten Lichtes ändert sich, wenn bei gleicher Einfalls- und Ausfallsrichtung die Fläche um die Normale gedreht wird (Textilien, gebürstete Metalle, Metalleffektlacke) - Superposition gilt, d.h. mehrere Quellen überlagern sich linear.

Es ist in der Computergrafik üblich, die bidirektionale Reflektivität als Gemisch von ambienten, diffusen und spekularen Komponenten  $d, s$  aufzufassen und einen ambienten Anteil  $a$  zu addieren. Für eine Menge Q von Lichtquellen berechnen wir damit die gesamte reflektierte Strahlstärke:

$$L_r = p_a * E_a + \sum_{1 \leq j \leq Q} E_j * (k_d * p_d + k_s * p_s) \text{ mit } k_d + k_s = 1 \text{ und } Q = \text{Anzahl der Lichtquellen}$$

Für ambiente und gerichtete Lichtquellen aus der Hemisphäre ergibt sich eine spezielle Form der BRDF, die Render-Gleichung (Jim Kajiya 1986): - eine BRDF mit Integral über alle Lichtquellen (bzw. Hemisphären) -

$$L_r = p_a + \int_{\Omega_{\text{Omega}}} L * (k_d * p_d + k_s * p_s) \omega_i * n d\Omega$$

- Ambiente Strahlung: - es ist keine "eigentliche" Quelle zuordenbar - stark vereinfachtes Modell für die Streuung der Atmosphäre, für viele "durchmischte" Strahlungsquellen, für indirekte Reflexionen - Strahlung kommt von allen Seiten "Die Quelle ist überall und nirgends keine Abhängigkeit von Winkeln und Entfernungen - Beschreibung nur indirekt durch konstante Bestrahlungsstärke (Irradiance) von Flächen möglich -  $E = \frac{\Phi}{A} = E_a$  - Parallele Strahlung: - Strahlung ist gerichtet und parallel (kollimiertes Licht, Strahlungsquelle im Unendlichen, Sonnenlicht) - für derartige Quellen lässt sich kein Ort (aber uneigentlicher Ort, Richtung) angeben - Wichtig sind die Richtung und die Strahlungsleistung, bezogen auf die senkrecht zur Ausbreitungsrichtung stehende Fläche (spezifische Ausstrahlung oder Radiosity  $R_e$ )  $R = E_q = \frac{\Phi}{A_q}$  - für die Schattierungsrechnung lässt sich die Bestrahlungsstärke  $E_e$  der Oberfläche (Flächenelement  $dA$ ) berechnen:

$$E = \frac{\Phi}{A} = \frac{E_q * A_q}{A} = E_q * \cos(\phi) = E_q * V_I^T * n$$

Ideale Punktlichtquelle: - für die Punktquelle ist der Ort bekannt und die Strahlstärke in alle Richtungen konstant:

$$I = \frac{\Phi}{\Omega} = \text{konstant} - \text{die Bestrahlungsstärke eines physikalischen vorliegenden, beliebig orientierten Flächenelementes A ergibt sich zu:}$$

$$E = \frac{\Phi}{A} = \frac{I * \Omega}{A}, \Omega = \frac{A}{r^2} * \cos(\phi) * \omega_r \rightarrow E = \frac{I}{r^2} * \cos(\phi) * \omega_r - \text{zum Ausgleich der Adaptionsfähigkeit des menschlichen Auges wird in der Computergrafik oft der folgende Ansatz verwendet:}$$

$$- E = \frac{I}{c_1 + c_2 * |r| + c_3 * r^2} * \cos(\phi) * \omega_r - \text{Remittierende Flächen}$$

(radiometrische Betrachtung): - Zur Berechnung der von der reflektierenden Fläche weitergegebenen Strahllichte L sind die weiter oben berechneten Bestrahlungsstärken E für die unterschiedlichen Quellen mit dem Faktor  $\frac{\beta(\lambda)}{\pi \omega_r}$  zu bewerten

— Quelle — Reflexion — Spektale Strahllichte  $L(\lambda)$  — — — — — ambient — diffus —  $L(\lambda) = \frac{E(\lambda)}{\pi \omega_r} * \beta(\lambda)$  — — —

gerichtet — diffus —  $L(\lambda) = \frac{E(\lambda)}{\pi \omega_r} * \cos(\phi) * \beta(\lambda)$  — —

punktförmig — diffus —  $L(\lambda) = \frac{I(\lambda)}{\pi r^2} * \cos(\phi) * \beta(\lambda)$  — —  
gerichtet diffus — diffus —

$L(\lambda) = \frac{I(\lambda)}{\pi r^2} * \cos^m(\theta) * \cos(\phi) * \beta(\lambda)$  —

Ein Beleuchtungsmodell ist eine Verfahren in der Computergrafik welches das Verhalten von Licht simuliert. Die Simulation unterscheidet dabei zwischen lokaler und globaler Beleuchtung: - Lokale Beleuchtungsmodelle: - simulieren das Verhalten von Licht auf den einzelnen Materialoberflächen - nur Beleuchtungseffekte welche direkt durch Lichtquellen auf einzelnen Objekt entstehen - indirekte Beleuchtung bleibt zunächst unberücksichtigt - Globale Beleuchtungsmodelle: - simulieren die Ausbreitung von Licht innerhalb der Szene - dabei wird die Wechselwirkung in der Szene beachtet (Schatttenwurf, Spiegelung, indirekte Beleuchtung)

- lokales Beleuchtungsmodell (lässt sich durch BRDF beschreiben) - eignet sich zur Darstellung von glatten, plastikähnlichen Oberflächen - baut nicht auf physikalischen Grundlagen auf - widerspricht dem Energieerhaltungssatz - Reflexion des Lichts = ambienter+ ideal diffuser + ideal spiegelnder Reflexion

- Allgemein:  $L = I_{out} = I_{ambient} + I_{diffus} + I_{specular}$  - Ambiente:  $I_{ambient} = I_a * k_a$  mit  $I_a$  Intensität des Lichtes und  $k_a$  Materialkonstante - Diffus:  $I_{diffus} = I_{in} * k_d * \cos(\phi)$  mit  $I_{in}$  Lichtstärke der Punktlichtquelle;  $k_d$  empirischem Reflexionsfaktor;  $\phi$  Winkel zwischen Oberflächennormale und Richtung des einfallenden Lichtstrahls - Spiegelnd:

$I_{specular} = I_{in} * k_s * \frac{n+2}{2\pi} * \cos^n(\theta)$  mit -  $I_{in}$  Lichtstärke des eingallendes Lichtstrahls der Punktlichtquelle -  $k_s$  empirisch bestimmter Reflexionsfaktor -  $\theta$  Winkel zwischen idealer Reflexionsrichtung des Lichtstrahls und Blickrichtung -  $n$  konstante Exponent zur Beschreibung der Oberflächenbeschaffenheit -  $\frac{n+2}{2\pi}$  Normalisierungsfaktor zur Helligkeitsregulierung - Vollständige Formel:

$I_{out} = I_a * k_a + I_{in} * k_d * \cos(\phi) + I_{in} * k_s * \frac{n+2}{2\pi} * \cos^n(\theta)$   
Unterschiedliche Definitionen sind möglich, z.B. mit mehrere Lichtquellen: - jeweiligen Komponenten für jede Lichtquelle separat berechnet - diese werden anschließend aufsummiert

- Physik-basierte spekulare Reflexion: - Microfacetten: Grundidee ähnlich Phong-Modell - Statistische Abweichung der Microfacetten von der Flächennormalen (z. B. Beckmann-Verteilung) - Streuung des Lichts (Keule) um den Winkel der idealen Spiegelung herum - Berücksichtigt auch die gegenseitigen Abschattung (insbesondere bei flachen Lichtstrahlen) - Vollständig physikbasiertes Modell, keine willkürlichen Reflexionskonstanten - Aufwendige Berechnung (verschiedene Näherungsformeln existieren) -

Beckmann-Verteilung:  $I_{spec} = \frac{\exp(-\frac{tan^2(\alpha)}{n^2})}{\pi m^2 \cos^4(\alpha)}$ ,

$\alpha = \arccos(N * H)$

Bisher: - Zerlegung gekrümmter Flächen in Polygone (meist Drei- oder Vierecke) - Positionen der (Eck-)Punkte und Normalen im 3D sowie der Punkte im 2D-Bild sind bekannt (per Matrixmultiplikation für Transformationen und Projektion) - Pixelpositionen für Polygone/Dreiecke im Bild

per Scanline-Algorithmus - lokale Beleuchtungsmodelle für 3D-Punkte (z.B. per Phong-Beleuchtungsmodell)

Jetzt: Wie kommt Farbe (effizient) in die Pixel? - Wie oft muss lokales Beleuchtungsmodell bei n Pixeln im Dreieck angewendet werden?

— Verfahren — Anz. — Idee — — — — — — — — — — —

Flat-Shading — 1 — eine Berechnung, dann gleiche Farbe für alle Pixel des Dreiecks/Polygons verwenden — —

Gouraud-Shading — 3 — pro Eckpunkt eine Farbe berechnen, dann lineare Interpolation (pro Dreieck) für jedes Pixel — — Phong-Shading — n — eine Berechnung pro Pixel, davor aber jeweils lineare Interpolation der Normalen pro Pixel —

→ Phong-Beleuchtungsmodell in jedem der obigen Shading-Verfahren nutzbar → hier nur direkte Schattierung (nur lokal, wo sind die Lichtquellen), d.h. nicht global (wie bei Radiosity & Raytracing)

Arbeitsweise des Flat-Shadings - stets nur 1 Farbwert pro (ebener) Fläche, - Stelle der Berechnung frei wählbar (möglichst repräsentativ), - repräsentativ wäre z.B.: Punkt (Ort mit Normale) in der Mitte der Fläche - → trivial für Drei- und Vierecke? → für Dreiecke und konvexe Vierecke!

Auswirkungen - "flaches" Aussehen und Helligkeitssprünge an den Kanten, das ist: - schlecht für Fotorealismus, - gut für abstraktere technische Darstellungen und - u.U. wichtig für realistische Darstellung kantiger Körper (insbes. wenn pro Eckpunkt nur eine Normale modelliert ist). - schneller als die anderen Verfahren, - u.U. genauso gut wie z.B.

Phong-Shading, wenn z.B.: - das Objekt sehr fein modelliert wurde oder - sehr weit entfernt ist - → d.h. nur ca. 1 Pixel pro Polygon/Dreieck gerendert wird (n==1)

- Gouraud-Shading [H. Gouraud 1971] schattiert Dreiecke (bzw. aus Dreiecken zusammengesetzte Polygone) kontinuierlich, - beseitigt damit die Diskontinuitäten des Flat-Shadings, - meist gleiche Normalen pro Vertex, d.h. pro Dreieck wirken oft 3 verschiedene Richtungsvektoren statt nur eine Normale (Dreiecksmittelpunkt) wie beim Flat-Shading und - lineare Interpolation der Schattierung (Intensitäten) im Inneren des Dreiecks aus den 3 Farbwerten der Eckpunkte. - Es werden "Normalenvektoren"  $n_i$  für jeden Eckpunkt  $P_i$  des Polygons ermittelt bzw. auslesen. - Die Herleitung der "Normalenvektoren"  $n_i$  ist aus der Originaloberfläche (z.B. Zylinder, Kegel, Bèzier-Fläche) oder Nachbarpolygone möglich. - Für jeden Eckpunkt: Berechnung der Beleuchtungsintensität  $I_i$  (z. B. nach dem Phong-Beleuchtungsmodell). - Normalen  $n_i$  der Eckpunkte werden entweder direkt aus den Flächen (z.B.

Regelgeometrien, bei Kugel z.B. Richtung des Radiusvektors) oder aus den Flächennormalen der benachbarten Polygone durch flächengewichtete Mittelung berechnet. - Die Schattierungsrechnung (RGB-Werte) erfolgt für die Eckpunkte und liefert die reflektierte Leuchtdichte  $I_i$ . Zur Erinnerung, das Phong-Beleuchtungsmodell: -

$I_{out} = I_a * k_a + I_{in} * k_d * \cos(\phi) + I_{in} * k_s * \frac{n+2}{2\pi} * \cos^n(\theta)$  -  $\cos(\phi) = V^T * n_i$ ,  $\cos^n(\theta) = (V^T * V_e)^n$  - Nach Anwendung des Beleuchtungsmodells an den Eckpunkten (auch Vertex-Shading genannt) - Bei der Rasterkonvertierung wird zwischen den

Eckwerte  $I_i$  linear interpoliert und damit die Intensität jedes Pixels der Rasterlinie berechnet (Intensität I steht hier für die Leuchtdichte oder für Farbwerte usw.) - Die Interpolation erfolgt nach dem gleichen arithmetischen Muster wie die Interpolation der x-Werte beim Polygonfüllalgorithmus, bzw. der  $1/z$ -Werte im z-Buffer-Verfahren (d. h. inkrementell, mit Ganzzahlarithmetik). - Für farbige Oberflächen werden die Leuchtdichten an den Polygonecken durch RGB-Werte beschrieben und ebenso zwischen den Ecken linear interpoliert. - Resultat: Kontinuierlich schattierte dreidimensionale Oberflächen

Artefakte des Gouraud-Shading, bedingt durch die lineare Interpolation: - Fehlen von gut ausgeprägten Glanzlichtern (verwischt oder verschwunden) - Mach-Band-Effekt: ((helle) Bänder) Kontrastverstärkung durch das Auge an den Übergängen zwischen Polygonen - Diese Artefakte werden im Folgenden genauer untersucht.

Auf Grund der linearen Interpolation von Intensitäten können Glanzlichter, die auf spekulare Reflexion zurückzuführen sind, verloren gehen oder abgeschwächt/verschmiert werden. Das wird umso kritischer, je spitzer die spekulare Reflexion ist (großes n im  $\cos^n$ - Term).

Feinere Unterteilung der Oberfläche verbessert Resultat Die lineare Interpolation der Leuchtdichte zwischen den Polygonkanten entlang der Rasterlinie führt zu einem Verlauf, der durch plötzliche Änderungen im Anstieg der Intensität gekennzeichnet ist (nicht stetig differenzierbar).

Der Mach-Band-Effekt: physiologisches Phänomen (Ernst Mach, 1865) - Bei Sprüngen in der Helligkeitsänderung (c0-Stetigkeit, c1-Unstetigkeit, typisch für Approximation durch ebene Polygone beim Gouraud-Shading, z.B. Zylinder) stört dieser Effekt u. U. erheblich. - Gleiche Information benachbarter Rezeptoren wirkt bei der weiteren visuellen Verarbeitung lateral hemmend auf die lokale Lichtempfindung. - Modellhaft entstehen neben dem eigentlichen Helleindruck auch "SSignale", die dem Helligkeitsgradienten (erste Ableitung) und dem Laplacefilter-Output (Laplacian of Gaussian / LoG, zweite Ableitung) entsprechen. - Die

Empfindung wird insgesamt nicht nur durch die Lichtintensität selbst, sondern auch durch die Überlagerung mit ihrer ersten und zweiten räumlichen Ableitung bestimmt. - Das führt zu einer Verstärkung von Konturen an "SSprungkanten" (c0-Unstetigkeiten, Intensitätssprünge).

In der dunklen Fläche zeigt sich eine dunklere, in den hellen Flächen eine hellere Kantenlinie. Dort, wo Konturen vorhanden sind, ist das vorteilhaft (evolutionäre Entwicklung der menschlichen visuellen Wahrnehmung), obwohl Täuschungen damit verbunden sind (photometrischer Eindruck).

- zunächst Kanten: Liegen eine helle und eine dunkle Fläche nebeneinander, beobachtet man einen dunklen Streifen auf der dunkleren Seite und einen hellen Streifen auf der helleren Seite (Kontrastverstärkung). - Bei einer Abfolge von Flächen unterschiedlicher Graufärbung, die in sich keine Farbgraduierung haben, beobachten wir entlang der Grenzen machsche Streifen (nach Ernst Mach 1865). Dabei handelt es sich um helle und dunkle Streifen, die den Kontrast zwischen den Flächen verstärken. [Quelle: Wikipedia]

Phong-Shading [Phong 1975]: - Lineare Interpolation der Normalenvektoren zwischen den Polygonecken anstelle von Interpolation der Intensitätswerte (bei Grafikkarten/-software als Pixelshader bekannt). - Exakte Berechnung der  $\cos^n$ -Funktion im Phong-Beleuchtungsmodell für jedes Pixel : Glanzlichter werden erhalten! - Keine Diskontinuität der ersten Ableitung: Mach-Band-Effekt wird vermieden!  
 Soll nur ein konvexes Objekt gerendert werden, dann ist die Entscheidung, welche Flächen zu zeichnen sind, einfach anhand der jeweiligen Normalen möglich.  
 Annahme: mehrere konvexe Objekte oder auch konkave Objekte sollen gerendert werden. Verdeckungen sind also möglich! - Korrekte Behandlung von Verdeckungen bedarf spezieller Ansätze/Datenstrukturen (Lösung des Reihenfolgeproblems). - Rein opake Szenen sind typischerweise wesentlich leichter zu implementieren als (teilweise) transparente (zusätzlich ein Berechnungsproblem). - Zeichenreihenfolge ist teilweise wichtig (z.B. von hinten nach vorn), - Algorithmen/Ansätze unterscheiden sich auch in der Granularität/Genauigkeit was auf einmal gezeichnet/sortiert wird: - Objekte (ganze Objekte nach z-Position sortieren, dann jeweils zeichnen...) - allg. (d.h. ggfs. überlappende) Polygone: Painters-Algorithmus, - überlappungsfreie Dreiecke/Polygone: Depth-Sort-Algorithmus, - Pixel: Z-Buffer-Verfahren (oft auch in Verbindung mit Obj.-Sort.) - Beliebte Testszene sind sich zyklisch überlappende Dreiecke, z.B.  
 - Gegeben sei eine 3D-Szene, bestehend aus grauen Polygonen mit diffus reflektierender Oberfläche, sowie eine gerichtete Lichtquelle. - Für jedes Polygon wird die reflektierte Strahlendichte L auf Basis des eingestrahlt Lichts (Richtung & Stärke) und der Flächennormale berechnet: -  $I_{out} = L = I_{in} * k_d * \cos(\phi)$  - Die Polygone werden mittels perspektivischer Kameratransformation (4 x 4 Matrix) in das Kamera-Koordinatensystem (Bildraum) transformiert und nach absteigendem z-Wert (Distanz des Polygonschwerpunkts zum Betrachter) sortiert. - Die sortierten Polygone werden der Reihe nach (entfernte zuerst) mit dem 2D-Polygonfüllalgorithmus in das Pixelraster der x/y-Bildebene konvertiert. - Die Pixel für jedes Polygon werden per Overwrite-Modus mit dem Farbwert L (nach obiger Berechnung) im Bildspeicher gespeichert. - Die Verdeckungsprobleme lösen sich durch die Reihenfolge quasi automatisch.  
 Gleichnis: Der Algorithmus arbeitet wie ein Maler, der zuerst den Hintergrund und dann Schritt für Schritt das jeweils weiter vorn liegende Objekt (oder Polygon bzw. Dreieck) zeichnet - und dabei die dahinterliegenden verdeckt. ABER, potentielle Probleme des Painter's-Algorithmus: selbst bei Dreiecken sind trotzdem falsche Verdeckungen möglich!  
 - Unterteilung in sich nicht überlappende und vollständig überdeckende Teilpolygone - Ist in der Projektionsebene durch gegenseitigen Schnitt aller Polygone möglich (allerdings blickabhängig - muss in jedem Bild neu berechnet werden!). - Die sichtbaren Teilpolygone können nun ausgegeben werden: - Zeichnen der nicht überlappenden Teilpolygone - Von den sich vollständig überlappenden Teilpolygenen wird nur das vordere gezeichnet.

- Eine einfache, nicht blickwinkelabhängige Unterteilung tut es in diesem Falle auch! - Die Teilpolygone sollten dabei möglichst nicht größer sein als der Tiefenunterschied, damit sie in jeder Situation eindeutig sortiert werden können! - Die 6 Teilpolygone können mittels Painter's Algorithmus korrekt sortiert und dargestellt werden  
 Anwendungsbereiche des Painter s Algorithmus / Depth-Sort Algorithmus: - Einfache Szenen, kleine Objekte, die sich in den z-Werten hinreichend unterscheiden. - Dort, wo keine Hardware-Unterstützung für 3D-Rendering angeboten wird (begrenzter Speicher, keine Z-Buffer Unterstützung). - Viele 2D-Grafiksystem bieten bereits Polygonfüllverfahren an. - Ähnliche Vorgehensweise wird auch für das Schattieren von semi-transparenten Flächen notwendig (s. später)!  
 Als Sortierverfahren für Echtzeitsysteme eignet sich z.B. "Insertion-Sort": - Begründung: Von Bild zu Bild ändert sich die Tiefenwerte (und damit die Reihenfolge) der Polygone relativ wenig. Damit sind die Polygone beim nächsten Bild bereits mehr oder weniger vorsortiert (nur wenige Polygone) müssen neu einsortiert werden. Die Komplexität von Insertion-Sort wird bei bereits sortierten Listen linear (O-Notation / best case). - Folglich tritt beim Painters-Algorithmus der best case sehr häufig ein (außer beim ersten Bild, wo man vom average case ausgehen kann- hier wird die Komplexität quadratisch).  
 - Einer der einfachsten "visible surface Algorithmen (CATMULL 1974) - Probleme des Painters-Algorithmus werden überwunden durch zusätzliche Berechnung des z-Wertes für jeden Punkt jedes Polygons und Speicherung des zur Projektionsebene nächstliegenden Farb- und Z-Wertes. - Dazu ist ein zusätzlicher Speicher (z-Buffer) für jedes Pixel notwendig. - Es sind weder Vorsortieren von Objekten noch Polygonzerlegung erforderlich (wenn alle Objekte opak sind).  
 Initialisierung: Für alle Pixel - Setze Farbe auf Hintergrundfarbe (z.B. Weiß) - Setze alle Z -Werte auf  $\infty$  (max. ganzzahliger Wert) - Setze Z min auf Wert der Near-Plane  
 Für alle Polygone (im 3D-Kamerakoordinatensystem) - Rasterumwandlung in der Projektionsebene ( $x_p/y_p$  Koordinaten) durch modifizierten 2D-Polygonfüllalgorithmus. Modifiziert heißt: zusätzliche Berechnung des z-Wertes für jedes Pixel - Anwendung einer Write Pixel ZB-Prozedur: - Wenn der z-Wert des aktuellen Pixels (im abzuarbeitenden Polygon) kleiner als der bereits abgespeicherte z-Wert ( $z_p$ ) an dieser Position ist, wird im z-Buffer bei  $x_p, y_p$  die Farbe sowie  $z_p$  ) überschrieben (mit den neuen Werten). - Sonst: alte Werte im Speicher bleiben erhalten - Die näher an der Kamera liegen Pixel überschreiben somit die weiter weg liegenden. - Pixelgenaue Sichtbarkeitsbestimmung und -behandlung der Polygone  
 Berechnen der z-Werte durch lineare Interpolation: - Die Tiefenwerte sind auch nach der Ansichten-Transformation (View-Transformation) zunächst nur für die Eckpunkte gegeben. - Zunächst erfolgt die lineare Interpolation der z-Werte entlang der Polygonkanten  $P_i P_j$  für die y-Position der gerade aktuellen Scanline - Danach wird mit dem Füllen der Bildzeile (z.B. durch einen konventionellen Polygonfüll-Algorithmus) die Interpolation der z-Werte

entsprechend der x-Position in der Scanline (Bildzeile) fortgesetzt (pixelgenaues Befüllen des z-Buffers).  
 Berechnung der z-Werte eines Pixels x/y: - Die y-Koordinate reicht zur Interpolation von  $z_A$  und  $z_B$  (Strahlensatz). - Pixel-z-Wert  $z_p$  wird äquivalent ermittelt, allerdings die Interpolationskoordinate jetzt x ( $y = \text{const}$  für die Rasterlinie) - Die Werte  $z_A, z_B, x_A, x_B$ , in  $z_p$  werden gleichzeitig mit den  $x_A$ -Werten (Schnitte) von einer Rasterlinie zur nächsten inkrementiert (s. Polygonfüllalgorithmus) - Die Brüche bleiben in allen Ausdrücken rational. - Die Ausdrücke für die z-Werte haben identische Form wie die der x-Werte beim Polygonfüllalgorithmus.  
 Immer Ganzzahlarithmetik! (ähnlich wie x-Werte im Polygonfüllalgorithmus)  
 Beispiel: Mögliche Berechnungen eines Tiefenwertes der Pixel - Als Beispiel dient hier eine Tischplatte (Rechteck, Größe 3m x 1m) in der Perspektive - Achtung: Eine lineare Interpolation der z-Werte im Bildraum (links) ist nicht wirklich korrekt! (höchstens als Näherung, OK für kleine nahe Flächen) -  $\frac{1}{z}$  kann exakt linear in x- & y-Richtung interpoliert werden (Abbildung rechts). - Da  $z_1$  abnimmt, wenn z zunimmt, muss aber der z-Test invertiert werden! - positive Auswirkung: Tiefeninfos naher Obj. werden mit höherer z-Genauigkeit gespeichert als weiter von der Kamera entfernte. Statistisch gesehen gibt es damit weniger  $\beta$ -Fighting“-Effekte (z.B. bei Bewegungen willkürliche Farbwechsel zwischen den Farben von Objekten mit nahezu der selben Tiefeninfo im z-Buffer).  
 - Das Ergebnis des Z-Buffer-Verfahrens ist vergleichbar mit dem Painters-Algorithmus. - Es ist jedoch bei opaken Objekten keine vorgängige Sortierung der Polygone nötig. Sie können in beliebiger Reihenfolge gezeichnet werden. - Die Interpolation der 1/z-Werte erfolgt im Polygonfüll-Algorithmus durch wenige Ganzzahl-Operationen (wie bei den x-Werten) - Das Verfahren ist pixelgenau: Es werden auch zyklisch sich überlappende (und sogar räumlich sich durchdringende) Polygone korrekt dargestellt. - Kaum Mehraufwand gegenüber dem 2D-Polygonfüllalgorithmus! - Mögliches Problem: Korrekte Berücksichtigung von Transparenzen!  
 Alpha-Blending-Verfahren: - Annahme: Verwendung eines Z-Buffers - Mit dem Alpha-Blending-Verfahren kann die transparente Überlagerung zweier Objekte im Bildspeicher wie folgt gelöst werden -  $C_f$  Farbe des Objekts im Vordergrund (kleinster z-Wert), -  $\alpha$  Opazität der Vordergrundfarbe, Wert zwischen 0 und 1 (bzw. 100-  $C_b$  Hintergrundfarbe (die im Bildspeicher für das entsprechende Pixel zuletzt eingetragene Farbe) - Die resultierende Farbe C ergibt sich zu:  
 $C = \alpha * C_f + (1 - \alpha) * C_b$  - Für Alpha-Blending wird der Bildspeicher (mit z-Buffer) um den Opazitätswert  $\alpha$  erweitert: - Speicherbedarf pro Pixel typischerweise mindestens 48 Bit: R + G + B + Z +  $\alpha$ . - Bei einer Auflösung des Bildschirms von 1.000.000 Pixel benötigen wir ca. 6MB Speicher. - z-Wert und  $\alpha$ -Wert des Vordergrund Objektes werden nach dem Alpha-Blending in den Bildspeicher übernommen!  
 - Reines Z-Buffering (ohne  $\alpha$ ) ignoriert alle Objektpixel, die weiter entfernt sind als vorn liegende Objektpixel (siehe rechts, hier ist die Reihenfolge egal). - Bei Berücksichtigung von

$\alpha$ -Werten (Transparenzen) ist die Renderreihenfolge für korrekte Ergebnisse aber sehr wichtig! (siehe Mitte bzw. links) - Erläuterung zum Transparenz-Problem: - Die Formel für  $\alpha$ -Blending berücksichtigt nur die Überlagerung des aktuellen Objektes mit dem davor existierenden Bildschirminhalt. Wird ein dazwischenliegendes Objekt nachträglich gezeichnet, dann kann die Farbe nicht korrekt bestimmt werden. Dies passiert aber beim Z-Buffering, da die Zeichenreihenfolge der Polygone beliebig ist. - **Im Beispiel** - Die opake grüne Kreisscheibe liegt zwischen dem hinteren Objekt (blau) und dem transparenten vorderen Objekt (rot), wird aber als letztes gerendert. → Grün kann Blau nicht mehr verdecken, denn Blau wurde zuvor schon mit Rot verrechnet (ist nun mit "vordererß-Koordinate im Z-Buffer hinterlegt). Dort, wo die grüne Kreisscheibe hinter dem transparenten Rot (bzw. dem nun Rot-Blau) liegt wird ein nicht korrekter Blauanteil gezeigt. Auch der weiße Hintergrund kann hinter dem transparenten Rot (insgesamt ein transparentes Rosa) nicht mehr vom Grün verdeckt werden! - algorithmische Lösung des Problems: - Zuerst: Darstellung aller opaken Objekte ( $\alpha = 1$ ) nach dem Z-Buffering (reihenfolgeunabhängig) - Dann Sortieren aller semitransparenten Polygone nach der Tiefe und Zeichnen nach dem Painters-Algorithmus unter Berücksichtigung des Z-Buffers mittels Alpha-Blending! - Restfehler: sich zyklisch überlappende oder sich durchdringende semi-transparente Flächen → exakte Behandlung durch die vorn beschriebenen Maßnahmen (Unterteilung der Polygone notwendig!) - BRDF: physikbasiertes, lokales Reflektionsmodell (Lichtquelle auf Material) → Funktion von Einfallswinkel, Betrachterwinkel, Wellenlänge (bzw. -breiche) - Rendergleichung (Kajiya) = BRDF, Integral über alle Lichtquellen (bzw. Hemisphäre) - Approximation durch lokales Phong-Beleuchtungsmodell → für "einfache" Materialien und Lichtquellen "korrekt genug direkte (lokale) Schattierungsverfahren (Flat-, Gouraud- und Phong-Shading) - Was noch fehlt: Interreflexionen zwischen Objekten... - globale Beleuchtung, d.h. jede Fläche kann als Lichtquelle dienen einfaches Ray-Tracing: Strahlenverfolgung, nicht rekursiv - Strahlen vom Augpunkt (Ursprung des Kamerakoordinatensystems) durch jedes Pixel des Rasters senden → keine Löcher - Schnittpunktberechnung mit allen Objekten → Schnittpunkt mit dem größtem z-Wert stammt vom sichtbaren Objekt - Strahlverfolgung (Anwendung des BRDF-Reziprozitätsprinzips) und Aufsummierung der (Lichtquellen-)Anteile aufgrund von material- und geometrieabhängigen Parametern (ggf. neben Reflexion auch Brechung) → Ergebnis: Helligkeits-/Farbwert pro Pixel - Bestimmung der diffusen und spekularen Lichtreflexion nach dem Phong-Beleuchtungsmodell - Bis hier nur einfache, lokale Beleuchtung (keine Spiegelung, Schatten, indirekte Beleuchtung)! → Vorzüge des RT kommen erst bei rekursivem Raytracing zum Tragen! - Berechnung von Sekundärstrahlen am Auftreffpunkt (Reflexions- und Schattenfühler) - Annäherung der Interreflexionen (mehrfache Reflexion zwischen den Objekten) durch ideale Spiegelung, d.h. Spiegelung des primären Strahls an  $\vec{n}$  im Auftreffpunkt und Erzeugung des sekundären Strahls

- beim Auftreffen des Strahls auf ein weiteres Objekt B Berechnung der diffusen und spekularen Reflexion der jeweiligen Lichtquelle (Schattenfühler, Phong-Modell) sowie Erzeugung eines weiteren Strahls durch ideale Spiegelung - Addition der Sekundärstrahlen an Objekt B zum Farbwert des Pixel am Objekt A (Anteil bei jeder weiteren Rekursion meistens fallend, da reflektierter Anteil bei jeder Reflexion abgeschwächt wird) → Rekursion kann abgebrochen werden, wenn Beitrag vernachlässigbar! Transparenz unter Berücksichtigung der Brechung beim Ray-Tracing: Richtung des gebrochenen Strahls berechnet sich aus dem Einfallswinkel zum Normalenvektor sowie den material- und wellenlängenabhängigen Brechungsindizes.

$$\eta_{e\lambda} * \sin(\theta_e) = \eta_{t\lambda} * \sin(\theta_t)$$

Beispiel Luft-Glas:

$$\eta_{\text{Luft, rot}} * \sin(\theta_{\text{Luft}}) = \eta_{\text{Glas, rot}} * \sin(\theta_{\text{Glas}}) \Rightarrow 1.0 * \sin(30) = 1.5 * \sin(\theta_{\text{Glas}}) \rightarrow \theta_{\text{Glas}} \approx \arcsin\left(\frac{\sin(30)}{1.5}\right) \approx 20$$

Die Farbe im betrachteten Punkt wird nicht durch die Farbe von Hintergrundobjekt B1 (wie im Fall nichtbrechender Transparenz) sondern durch die Farbe von B2 beeinflusst! Berechnung des Einheitsvektors  $\vec{V}_t(\vec{V}_e, n, \theta_t)$  in Richtung der Brechung: - An Grenzflächen mit unterschiedlichen Brechungsindizes tritt neben der Transparenz ( $\vec{V}_t$ ) auch Reflexion (Komponente mit der Richtung  $\vec{V}_r$ ) auf. -  $\vec{M}$  ist ein Einheitsvektor (Länge=1) mit der Richtung von  $\vec{n} * \cos(\theta_e) - \vec{V}_e$  und - es gilt:

$$\vec{M} * \sin(\theta_e) = \vec{n} * \cos(\theta_e) - \vec{V}_e \rightarrow \vec{M} = \frac{\vec{n} * \cos(\theta_e) - \vec{V}_e}{\sin(\theta_e)}$$

- Effekte an transparentem Material: - Simulation brechungsbedingter Verzerrungen wird so möglich (z.B. bei optischen Linsen, Wasser). - Transparentes und reflektierendes Material erzeugt 2 weiter zu verfolgende Sekundärstrahlen.

Unzulänglichkeiten des einfachen rekursiven Ansatzes: - Reale Objekte sind eher diffus spekulär, d.h. ein ganzes Set von Sekundärstrahlen wäre zu verfolgen. - Die ideale Spiegelung zur Erzeugung von Sekundärstrahlen ist eine sehr starke Vereinfachung - Aus der Umkehrbarkeit von Licht- und Beleuchtungsrichtung ließe sich eine Menge von Sekundärstrahlen aus dem Phong-Modell ( $\cos^n(\theta)$ -Term) ermitteln. - Aus Aufwandsgründen (rein theoretisch wären unendlich viele Sekundärstrahlen zu berücksichtigen) muss vereinfacht werden, z.B. Monte-Carlo-Ray-Tracing **Monte Carlo Ray-Tracing**: - Reflexion ist selten ideal spekulär, meist entsteht ein Bündel von Strahlen - Ansatz: Verfolgung mehrerer zufälliger Sekundärstrahlen, deren Beitrag zum Farbwert des Pixel statistisch gewichtet wird. - Je gestreuter die Reflexion, um so mehr Sekundärstrahlen sind nötig. Sehr breite Remissionskeulen oder gar diffuse Interreflexionen sind wegen des Aufwandes nicht (bzw. nur schwer) behandelbar.

Beleuchtungsphänomen Kaustik: - Das Licht der Lichtquelle werde zuerst spekulär, dann diffus reflektiert. Beispiel: Lichtstrahlen, die von Wasserwellen reflektiert auf eine diffuse Wand auftreffen. - Vom Auge bzw. Pixel ausgehendes Ray Tracing versagt wegen des vorzeitigen Abbruchs der Rekursion

am diffus remittierenden Objekt. - Inverses Ray Tracing [Watt/Watt 1992] : Man erzeugt einen von der Lichtquelle ausgehenden Strahl und reflektiert diesen an glänzenden Oberflächen. Auch Photon Mapping kann hier helfen. - Die reflektierten Lichtstrahlen wirken als zusätzliche Lichtquellen, die dann zu diffusen Reflexionen führen können. Optimierungsmöglichkeiten (einfache Hüllgeometrien, Raumzerlegung, ...): - Berechnung von achsenparallelen Hüllquadraten (Bounding Boxes) oder Hüllkugeln (Bounding Spheres) um Objekte aus mehreren Polygonen. - Zunächst Test, ob der Strahl die Hülle schneidet und falls ja - → Schnittpunktberechnung von Strahl mit allen Polygonen in der Hülle - → zunächst Berechnung des Schnittpunktes mit der jeweiligen Polygonebene - → danach effizienter Punkt-im-Polygon-Test - Effiziente Zugriffsstruktur auf die Hüllquader: Bäume für rekursive Zerlegungen des 3D-Raumes (Octrees), Binary-Space-Partition-Trees - Verwendung von direktem, hardware-unterstützten Rendering (z.B. Gouraud- oder Phong-Shading) anstelle von einfachem, nichtrekursivem Ray-Tracing, nur bei Bedarf Erzeugung von Sekundärstrahlen. - Verwendung von Hardware mit RTX-Unterstützung Anwendung: - Erzeugung realistischerer Bilder als bei lokalem Shading, da indirekte (spekuläre) Beleuchtungsphänomene physikalisch (geometr. und radiometr.) viel genauer als bei direkter Schattierung berechnet werden können. - Ray-Tracing ist aufgrund der hohen Komplexität für interaktive Anwendungen (oft noch) wenig geeignet (hardware- und szenenabhängig), mögliche Lösung: Vorberechnung der Bildsequenzen im Stapel-Betrieb (batch mode) - Fotorealistic Visualisieren (Designstudien usw.) - Computeranimation in Filmen - Interaktive Programme (CAD, Spiele) verwenden noch eher direktes Rendering mit Texturen (shadow map, environment map) um Schatten, Spiegeleffekte oder Brechung zu simulieren. - Aufwendige Teiloperation: Geometrischer Schnitt im Raum: - für jedes Pixel: Berechnung des Schnittes eines Strahles mit potentiell allen Objekten der Szene (einfaches Ray-Tracing, ohne Rekursion) - z.B. Bildschirm mit 1.000 x 1.000 Pixeln und 1.000 Objekten - **Rekursives Ray-Tracing** für den ideal spiegelnden Fall: Anzahl der Operationen wächst zusätzlich, d.h. Multiplikation des Aufwandes mit der Anzahl der Reflexionen und Refraktionen und Lichtquellen (Schattenfühler) → für ca. 4 Rekursionsstufen bei 2 Lichtquellen haben wir etwa  $4 * (2 + 1) = 12$  Millionen Strahlen, was schon bei 1.000 Objekten 12 Milliarden Schnittpunktberechnungen bedeutet. - **Monte-Carlo-Ray-Tracing** für die Approximation diffuser Anteile: Weiteres Anwachsen der Anzahl an erforderlichen Operationen durch zusätzliche Verfolgung sehr vieler Sekundärstrahlen (durchschnittlich 10 pro Reflexion) → Mehrere 100 Millionen bis Milliarden Strahlen (bzw. Billionen Schnittpunktberechnungen) - Durch **effiziente räumliche Suchstrukturen** kann die Anzahl der tatsächlich auszuführenden Schnittpunktberechnungen wesentlich reduziert werden. Die Anzahl der Schnitte steigt nicht mehr linear (sondern etwa logarithmisch) mit der Anzahl der Objekte (siehe räumliche Datenstrukturen). Damit ist auch bei großen Szenen nur noch die Anzahl der Strahlen wesentlich →

je nach Bildauflösung und Verfahren, mehrere Millionen bis Milliarden Strahlen! - Eigenschaften des Ray-Tracing-Verfahrens: - Implementierung ist konzeptionell einfach + einfach parallelisierbar. - Hohe Komplexität durch Vielzahl der Strahlen, deshalb meistens Beschränkung auf wenige Rekursionen. - Exponentielle Komplexität bei Monte-Carlo-Ray-Tracing bzw. wenn alle Objekte gleichzeitig transparent (Brechung) und reflektierend sind. - Resultat: - RT ist sehr gut geeignet, wenn die spiegelnde Reflexion zwischen Objekten (und/oder die Brechung bei transparenten Objekten) frei von Streuung ist. - Die diffuse Reflexion zwischen Objekten wird beim Ray-Tracing durch ambiente Terme berücksichtigt. Eine bessere Beschreibung dieser Zusammenhänge ist mit Modellen der Thermodynamik möglich. - Weitere Ansätze: - Cone-Tracing - statt eines Strahles wird ein Kegel verwendet, der die Lichtverteilung annähert [Watt/Watt 1992]. - Radiosity (siehe Abschnitt weiter unten) - Photon Mapping (nächster Abschnitt) - Verfahren von Henrik Wann Jensen 1995 veröffentlicht - angelehnt an Teichencharakter des Lichts - 2-stufiges Verfahren - Quelle: Vorlesung von Zack Waters, Worcester Polytechnic Inst.

1. Phase: Erzeugung der Photon Map 1. Photonenverteilung in der Szene: Von der Lichtquelle ausgestrahlte Photonen werden zufällig in der Szene gestreut. Wenn ein Photon eine Oberfläche trifft, kann ein Teil der Energie absorbiert, reflektiert oder gebrochen werden. 2. Speichern der Photonen in der Photon Map Daten enthalten also u.a. Position und Richtung beim Auftreffen sowie Energie für die Farbkanäle R,G,B - Photon wird in 3D-Suchstruktur (kd-Baum) gespeichert (Irradiance cache) - Reflektionskoeffizienten als Maß für Reflektionswahrscheinlichkeit (analog Transmissionswahrscheinlichkeit) - dafür: Energie bleibt nach Reflexion unverändert. Neue Richtung wird statistisch auf Basis der BRDF gewählt. 2. Phase: Aufsammeln der Photonen aus Betrachtersicht (gathering) - Verwende Ray-Tracing um für den Primärstrahl von der Kamera durch einen Pixel den Schnittpunkt x mit der Szene zu bestimmen. Basierend auf den Informationen aus der Photon Map werden für x folgende Schritte ausgeführt: 1. Sammle die nächsten N Photonen um x herum auf durch Nächste-Nachbar-Suche in der Photon Map (N = konst., z. B. 10) 2. S sei die (kleinste) Kugel, welche die N Photonen enthält. 3. Für alle Photonen: dividiere die Summe der Energie der gesammelten Photonen durch die Fläche von S (→ Irradiance) und multipliziere mit der BRDF angewendet auf das Photon. 4. Dies ergibt die reflektierte Strahldichte, welche von der Oberfläche (an der Stelle x) in Richtung des Beobachters abgestrahlt wird.

Grundprinzip des Radiosity-Verfahrens: - Ansatz: Erhaltung der Lichtenergie in einer geschlossenen Umgebung - Die Energierate, die eine Oberfläche verlässt, wird Radiosity (spezifische Ausstrahlung) genannt. - Die gesamte Energie, die von einer Oberfläche (Patch, Polygon) emittiert oder reflektiert wird, ergibt sich aus Reflexionen oder Absorptionen anderer Oberflächen (Patches, Polygone). - Es erfolgt keine getrennte Behandlung von Lichtquellen und beleuchteten Flächen, d.h. alle Lichtquellen werden als emittierende Flächen

modelliert. - Da nur diffuse Strahler (Lambertstrahler) betrachtet werden, herrscht Unabhängigkeit der Strahldichte vom Blickwinkel vor. - Die Lichtinteraktionen werden im 3D-Objektraum (ohne Berücksichtigung der Kamera) berechnet. - Danach lassen sich beliebig viele Ansichten schnell ermitteln (Ansichtstransformation, perspektivische Projektion, Verdeckungsproblematik, Interpolation). Die gesamte von Patch  $A_s$  stammende Strahldichte an der Stelle von  $dA_r$  ist:  

$$L_r = \beta_r(\lambda) * \int_{A_s} \frac{L_s}{\pi * r^2} * \cos(\theta_s) * \cos(\theta_r) * dA_s$$
 (s=Sender, r=Reveiver)  
 Für das Polygon  $A_r$  ist die mittlere Strahldichte zu ermitteln!

$$L_r = \beta_r(\lambda) * \frac{1}{A_r} * \int_{A_r} \int_{A_s} \frac{L_s}{\pi * r^2} * \cos(\theta_s) * \cos(\theta_r) * dA_s * dA_r$$

Die Geometrieanteile aus dieser Gleichung werden als Formfaktoren bezeichnet (+Sichtbarkeitsfaktor  $H_{sr}$ ).

$$F_{sr} = \frac{1}{A_r} \int_{A_r} \int_{A_s} \frac{\cos(\theta_s) * \cos(\theta_r)}{\pi * r^2} * H_{sr} * dA_s * dA_r, H_{sr} = \begin{cases} 1 & \rightarrow \text{Sichtbar} \\ 0 & \rightarrow \text{nicht sichtbar} \end{cases}$$

Für Flächen, die klein im Verhältnis zu ihrem Abstand sind, ergibt sich eine Vereinfachung des Formfaktors. In diesem Fall können die Winkel  $\theta_s, \theta_r$  und Radius r über den zu integrierenden Flächen als konstant (Mittelwerte) angenommen werden.

$$F_{sr} = A_S \frac{\cos(\theta_s) * \cos(\theta_r)}{\pi * r^2} * H_{sr}$$

Bei dicht benachbarten Flächen gelten die obigen, vereinfachenden Annahmen u.U. nicht mehr. Es müsste exakt gerechnet oder in diesen Bereichen feiner untergliedert werden. Wird statt  $\beta\lambda$  vereinfachend ein konstanter Remissionsfaktor R (R diff im monochromatischen Fall oder  $R_{diffR}, R_{diffG}, R_{diffB}$  für die drei typischen Farbkanäle) eingeführt, so ergibt sich zwischen der Strahldichte  $L_r$  der bestrahlten Fläche und der Strahldichte  $L_s$  der bestrahlenden Fläche der folgende Zusammenhang:  $L_r = R_r * F_{sr} * L_s$  Jedes Patch wird nun als opaker Lambertischer (d.h. ideal diffuser) Emitter und Reflektor betrachtet (d.h. alle Lichtquellen werden genauso wie einfache remittierende Flächen behandelt, allerdings mit emittierendem Strahldichte-Term  $L_{emr}$ ).  $L_r = L_{emr} + R_r * \sum_S F_{sr} * L_s$  Es ergibt sich schließlich als Gleichungssystem:

$$\begin{pmatrix} 1 - R_1 F_{11} & -R_1 F_{12} & \dots & -R_1 F_{1n} \\ 1 - R_2 F_{21} & -R_2 F_{22} & \dots & -R_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 - R_n F_{n1} & -R_n F_{n2} & \dots & -R_n F_{nn} \end{pmatrix} * \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_n \end{pmatrix} = \begin{pmatrix} L_{em1} \\ L_{em2} \\ \vdots \\ L_{emn} \end{pmatrix}$$

Das Gleichungssystem ist für jedes Wellenlängenband, das im Beleuchtungsmodell betrachtet wird, zu lösen ( $R_r, R_{rR}, R_{rG}, R_{rB}, L_{emr}$  sind im Allgemeinen wellenlängenabhängig). Adaptives Radiosity-Verfahren: - vereinfachte Formfaktor-Berechnung ist ungenau bei eng beieinander liegenden Flächenstücken (z. B. in der Nähe von Raumecken),

oder bei kontrastreichen Übergängen) - deshalb adaptive Unterteilung solcher Flächen in feinere Polygone Im adaptiven Radiosity-Verfahren werden deshalb große Flächen (insbesondere dort wo Flächen relativ hell sind im Vergleich zur Nachbarfläche → kontrastreiche Übergänge) unterteilt. Die Notwendigkeit wird durch erste Berechnung mit grober Unterteilung geschätzt. - das Radiosity-Verfahren ist sehr aufwendig (Bestimmung aller Formfaktoren, Anwendung des Gauß-Seidel-Verfahrens zum Lösen des Gleichungssystems) - jedoch viel weniger Samples als Monte-Carlo-Raytracing (1 mal pro Kachel-Paar mal Anzahl Iterationen)! - beim progressive Refinement ist die inkrementelle Approximation des Ergebnisses des exakten Algorithmus durch ein vereinfachtes Verfahren wünschenswert - ein entsprechender Algorithmus, der die Patches einzeln behandelt, stammt von Cohen, Chen, Wallace und Greenberg - akkumuliert mehr Energie in jedem Schritt, verletzt Gleichgewicht der Strahlung → Korrektur notwendig:  $L_s^{k+1} = L_s^k + R_r * \sum_S F_{sr} * L_s^k$  - ausschließlich Berücksichtigung der diffusen Reflexion - blickwinkelunabhängig, direkt im 3D-Raum arbeitend - realistische Schattenbilder, insbesondere Halbschatten (viele, bzw. flächig ausgedehnte Lichtquellen) - sehr rechenintensiv, deshalb meist Vorausberechnung einer Szene in 3D → Beleuchtungsphänomene wie z.B. indirektes Licht (besonders augenfällig in Innenräumen, Museen, Kirchen, Theaterbühnen usw.) sind mit Radiosity sehr gut/realistisch darstellbar. - → die Kombination von Radiosity und Ray Tracing (und ggfs. anderen Verfahren/Filtern etc) ermöglicht computergenerierte Szenen mit sehr hohem Grad an Realismus. - BRDF für physikbasierte, lokale Berechnung der Reflexion von Lichtquellen als Funktion von Einfallswinkel und Betrachterwinkel (evtl. wellenlängenabhängig, oder einfach durch RGB) - Rendergleichung (Kajiya) = BRDF, Integral über alle Lichtquellen (bzw. Hemisphäre) - für indirekte Beleuchtung / Global Illumination: (verschiedene algorithmische Verfahren unter Verwendung der lokalen Beleuchtung (BRDF) - (rekursives) Raytracing (einfache Spiegelung, Brechung, Schatten) - Monte Carlo RT, (gestreute Spiegelung, diffuse Reflexion), Backward Ray Tracing (Kauistik), Photon Mapping → jedoch extrem rechenaufwendig!) - Radiosity (indirekte diffuse Reflexion - sichtenunabhängige Vorausberechnung in 3D für statische Szenen) - verschiedene Verfahren können kombiniert werden um die globale Beleuchtungsphänomene effizienter zu berechnen. - z. B. Radiosity + Ray Tracing: Indirekte diffuse Beleuchtung + Spiegelung und Schatten, etc. - typische Anwendung: Applizieren von 2D-Rasterbildern auf 3D-Modellen - Beispiele: Hausfassade, Holz-, Marmor-, Steintexturen, Tapeten, Stoffe etc. - 3D-Objekte mit relativ einfachen Polygonen modelliert. - Details als Texturen, (d.h. als Raster-Bilder) - gelegentlich "Impostor" genannt. - Texture-Mapping als Erweiterung des einfachen Pattern-Filling (siehe. Polygonfüllalgorithmus) - als Verallgemeinerung auch Image-based Rendering genannt - Verwendung unterschiedlicher 3D-Transformationen und Beleuchtungsarten - Spezielle Effekte! (Reflexionen, Schatten, ..)

Erzeugung von Texturen: - "reale" Texturen aus realen rasterisierten/digitalen Fotografien (aus Pixeln = "Picture-Elementen" werden Texel = "Texturelemente") vs. - "berechnete" Texturen → synthetische Computergrafik-Bilder: - vorberechnete reguläre Texturen (basieren auf Texeln) vs. - nach Bedarf erzeugte statistische bzw. prozedurale Texturen (Absamplen von mathematischen Beschreibungen, ggf. beliebig genau)

Anwendung von Texturen - Grundprinzipien: - Transformation des Texturraums in den Bildraum der Darstellung:

Verwendung unterschiedlicher geometrischer Transformationen (je nach Anwendungszweck) - Resampling: transformiertes Texturraster wird aufs Bildraster "gerundet" Filtern:

Verhindern/Abmildern von resampling-basierten Aliasing-Effekten - Beleuchtung: RGB-Werte der Textur dienen als Materialattribute bei der Beleuchtungsrechnung Unterschiedliche Arten des Texturmappings

(Transformationsfunktion): - Parametrisches Mapping: Ein Rasterbild wird auf ein 3D-Polygon aufgebracht, indem man den Eckpunkten (x,y,z) des Polygons 2D-Texturkoordinaten (u,v) explizit zuordnet. - affines Texturmapping: direkte affine Abbildung der Textur auf projizierte Polygone im Bildraum - perspektivisches Texturmapping: Zwischenabbildung der Textur in den 3D-Objektraum und perspektivische Projektion in den Bildraum - Projektives Texturmapping: Verwendung unterschiedlicher Projektionsarten (parallel, perspektivisch, eben, zylindrisch, sphärisch) - Environment-Mapping: Spiegelung der Textur an der Oberfläche (bzw. Refraktion) mit entsprechender Verzerrung - Transformation abhängig von Kameraposition!

Durch Zuordnung von 3 Punkten im Bildraster zu den entsprechenden 3 Punkten im Texturraster erhält man ein Gleichungssystem mit 6 Gleichungen und 6 Unbekannten ( $a_u, b_u, c_u, a_v, b_v, c_v$ ): -

$$P_1 : u_1 = a_u * x_1 + b_u * y_1 + c_u; v_1 = a_v * x_1 + b_v * y_1 + c_v -$$

$$P_2 : u_2 = a_u * x_2 + b_u * y_2 + c_u; v_2 = a_v * x_2 + b_v * y_2 + c_v -$$

$$P_3 : u_3 = a_u * x_3 + b_u * y_3 + c_u; v_3 = a_v * x_3 + b_v * y_3 + c_v$$

Für jedes Pixel(x,y) im Polygon: Resampling der Textur(u,v) bei der Rasterkonvertierung (Polygonfüllalgorithmus)

Für jedes Pixel(x,y) finde die Texturkoordinaten(u,v), d.h.: - Rückwärtstransformation vom Ziel zum Original → keine Löcher im Bild! - ABER: Texturkoordinaten können überspringen oder wiederholt werden! - Störsignale (Aliasing) → Filterung notwendig!

Affines Mapping der Vertices x,y auf u,v → lineare Interpolation der u/v-Texturkoordinaten zwischen den Vertices für jedes Pixel (ähnlich wie RGB- bzw. Z-Werte im Polygonfüllalgorithmus, durch Ganzzahlarithmetik) Problem: Durch affine 2D-Abbildungen können nur Transformationen wie Rotation, Skalierung, Translation, Scherung in der Bild-Ebene abgebildet werden, aber keine Perspektive! → Abbildungsfehler zwischen den Eckpunkten! (kleine Dreiecke → kleiner Fehler!)

Beispiel: affine 3D-Abbildung der Textur per 4x4-Matrix auf 3D-Modell: Texturraum → Objektraum: Rotation, Translation, Skalierung (...) dann Objektraum → Bildraum: Projektion (selbe wie bei Geometrieprojektion)

entspricht affinem Textur-Mapping mit einem zusätzlichen Zwischenschritt, der Bestimmung der Objektraumkoordinaten: - Matrix  $M_{to}$ : Koordinatentransformation vom Texturraum in den 3D- Objektraum (affine Abb.: 3D-Translation, -Rotation, -Skalierung) - Matrix  $M_{oi}$ : Koordinatentransformation vom Objektraum in den Bildraum (Kameratransformation, perspektivische Abbildung) - Matrix  $M_{ti}$ : gesamte Koordinatentransformation vom Texturraum direkt in den Bildraum:  $M_{ti} = M_{to} * M_{oi}$  - Matrix  $M_{ti}^{-1}$ : Inverse Koordinatentransformation vom Bildraum zurück in den Texturraum

→ 4x4-Matrix für homogene Koordinaten. Perspektivische Abbildung im Bildraum durch Division durch z, für jedes Pixel (wesentlich aufwendiger als lineare Interpolation)

Vergleich: Perspektivisches / Affines Texture Mapping: - perspektivisches Textur-Mapping liefert auch bei perspektivischer Ansicht geometrisch korrekte Bilder - etwas höherer Berechnungsaufwand pro Polygon, da für jedes Polygon zwei Transformationsmatrizen und eine inverse 4x4-Matrix bestimmt werden müssen - wesentlich höherer Berechnungsaufwand pro Pixel: Matrixmultiplikation plus (floating-point) Division! - bei affinem Textur-Mapping können hingegen einfach die Texturkoordinaten (u/v) zwischen den Polygonecken linear interpoliert werden: - ähnlich wie bei anderen Attributen (z. B. x-Koordinate (s. Edge-Scan), r/g/b-Werte (s. Gouraud-Shading), Tiefenwerte (1/z) funktioniert dies inkrementell und mit Ganzzahlarithmetik (als Teil des Polygonfüllalgorithmus) - je kleiner die Polygone im Bild, desto kleiner der Fehler beim affinen Texturmapping (Ansatz: feinere Unterteilung der Polygone in kleinere Dreiecke → dafür jedoch mehr Polygone!)

Eingliederung in die Render Pipeline - Bestimmung der zum Polygon gehörenden sichtbaren Pixel im Bildraum (Polygonfüllalgorithmus) - Ermittlung der zur jeder Pixelkoordinate gehörenden Texturkoordinate mit Hilfe der inversen Transformationsmatrix  $M_{ti}^{-1}$  - Ermittlung der Farbe des zu setzenden Pixels aus dem Texturraster (und gegebenenfalls weitere Schattierung aus der Beleuchtungsrechnung) - Beleuchtungsrechnung, z.B.: Multiplikation der Helligkeit einer beleuchteten diffusen weißen Oberfläche mit den r/g/b-Werten der Textur (Lambert Modell)

- Texturen mit Transparenz: RGBA-Wert zu jedem Pixel gespeichert, d.h. beim Rendern wird Alpha Blending mit der Hintergrundfarbe angewendet - Video Texture: zeitlich veränderliche Textur, d.h. dynamische Veränderungen wie z.B. Feuer, Rauch (mit Alpha-Blending über Hintergrund / Billboard) oder Fernseher in der Wohnung mit Programm" (ohne Alpha-Blending) - Solid Textures: - Textur als 3D-Array (u/v/w-Koordinaten, bzw. Voxel) → gespeicherte RGB(A)-Werte pro Voxel - Abbildung über affine 3D-Transformation xyz auf uvw - beim Rendern entweder auf Vertices angewendet und dann für Pixel linear interpoliert oder für jedes Pixel einzeln angewendet (Pixelshader) - Anwendungsbsp.: Schnitt durch Material (z.B. Massivholz, Marmor) oder Volume Rendering (Überlagerung von Schichten) mit Alpha Blending, z.B. Computertomographie

(CT-Daten) - ggfs. auch Videotextur als Spezialfall einer Solid Texture: Zeit als 3. Dim.

Berechnung der Texturkoordinaten aus der aktuellen Position der einzelnen Polygone (Analogie: Projektion eines Diapositivs auf ein räumliches Objekt)

Beispiel: Parallelprojektion mit fixer Position des Projektors zum Objekt - 2D-Textur (Bsp. Gitter aus Millimeterskalen) - Parallelprojektion der Textur auf einen Zylinder mit abgeschrägten Endflächen - Projektion ist relativ zum Objekt definiert, d.h. die Textur bewegt sich mit dem Körper, sofern man diesen bewegt - markierte Bereiche (1 bzw. 2) haben auf Zylinder stets identische Positionen - keine explizite Zuordnung von uv-Koordinaten zu Polygoneckpunkten notwendig, weniger Modellieraufwand!

Anwendungsbeispiele für projektives Textur-Mapping (Parallel- oder Zentralprojektion): - Darstellung geometrischer Eigenschaften (geometrische Details, parallel, fixe Position des Projektors zum Objekt, senkrecht zur Fläche) - einfache Darstellung von Parameterlinien (sofern die Textur senkrecht auf die Projektionsebene projiziert wird, parallel, fixiert bezgl. Objekt) - Simulation eines Lichtkegels (Repräsentation der Leuchtdichteverteilung der Lichtquelle (Lichtfeld) als Rasterbild in einer Textur, zentral, fix in Weltkoordinaten) Zylindrisches Textur-Mapping: - radiale Projektion der Textur-Koordinaten auf eine Zylinderoberfläche - visueller Effekt für zylinderähnliche Objekte ähnlich zu parametrischem Textur-Mapping, z.B. Etikett auf Flasche, Dose, etc. Sphärisches Textur-Mapping: - Zentralprojektion der Textur-Koordinaten auf eine Kugeloberfläche - Vorteil des projektiven Texturmappings: Eine explizite Zuordnung der 3D-Punkte zu Texturkoordinaten mit stetiger Fortsetzung der Parametrisierung an den Polygongrenzen entfällt → weniger Modellieraufwand!

Spezialfall des projektiven Textur-Mapping: - Simulation der Reflexion der Umgebung an einer reflektierenden Fläche - Darstellung abhängig von der Position des Betrachters sowie von den Normalen der reflektierenden Fläche - Textur entspricht der Lichtquelle für die Beleuchtung durch die Umgebung (Environment Map): Sphere Map bzw. Cube Map Mapping der Textur auf die spiegelnde Oberfläche: - Aussenden eines Strahls vom Auge auf einen Punkt der spiegelnden Oberfläche - Ermittlung der Reflexionsrichtung entsprechend dem Einfallswinkel des Strahl zur Flächennormale - damit Bestimmung des zu reflektierenden Punktes in der Umgebung, d. h. des entsprechenden Textur-Pixels aus der Environment Map Grundannahme beim Environment Mapping: - relativ große Entfernung der reflektierten Objekte von der spiegelnden Fläche

Erzeugung einer Cube Map-Textur: - Aufteilung der Environment Map in sechs Bereiche, die den sechs Flächen eines Würfels um die spiegelnde Fläche herum entsprechen - Rendern der Umgebung sechs mal mit einem Kamera-Sichtfeld von jeweils 90 Grad aus dem Mittelpunkt des Würfels - Alternativ: Digitale Aufnahme und Einpassen der sechs Flächen mittels Image Warping in die jeweiligen Zonen der Environment Map - Strahlverfolgung: Sehstrahl wird an den

Eckpunkten des Objekts (entsprechend den Normalen) gespiegelt und dreidimensional mit den 6 Wänden der Cube Map geschnitten. - Daraus ergibt sich eine Zuordnung von Objektkoordinaten ( $x/y/z$ ) und Texturkoordinaten ( $u/v$ ). - Die Transformation kann wie beim perspektivischen Texturmapping berechnet werden und beim Rasterisieren für die dazwischen liegenden Pixel angewendet werden. - Effekt ähnlich wie bei Raytracing, jedoch geometrisch angenähert (gespiegelte Objekte sind nur als 2D-Raster-Bild repräsentiert) - keine aufwändigen Strahl-Objektschnitte (wie beim Raytracing) notwendig (Sehstrahl wird von den dargestellten Dreiecksseiten zurückgerechnet!) - Näherung wird ungenau, wenn das spiegelnde Objekt weit weg ist von der Kameraposition, welche für die Generierung der Cube-Map verwendet wurde - nur Einfachreflexion - Cube Maps können dynamisch (durch Offline-Rendering in Texturbuffer) generiert werden. Dadurch auch bewegte gespiegelte Objekte in Echtzeit darstellbar - Beachte: gespiegeltes Dreieck kann auf zwei oder mehrere Wände der Cube Map fallen. Dies kann durch mehrfaches Mapping und Clipping gelöst werden.

Environment Mapping [Haerberli/Segal 1993] für Kugel und Torus: - Unterschiedliche Ausrichtung der Objektfläche sorgt für korrekte Verzerrung der spiegelnden Objekte. Die Darstellung der spiegelnden Objekte (Geometrie und Material) steht beim Environment-Mapping im Vordergrund und nicht die korrekte geom. Darstellung gespiegelter Objekte! - Alle Raumrichtungen werden auf der Kugeloberfläche abgebildet. Je nach Aufnahmegeometrie mehr oder weniger großer blinder Fleck“ hinter der Kugel.

Erstellung einer Spherical-Environment-Map-Textur: - spiegelnde Kugel in der Mitte einer Szene - Fotografie der Kugel mit einer Kamera sehr großer (unendlicher) Brennweite aus großem (unendlichem) Abstand (parallele Projektionsstrahlen) - Entstehung einer kreisförmigen Region in der Textur-Map mit den Tangenten jeweils an den Außenkanten - Texturwerte außerhalb des Kreises werden nicht benötigt - Wahl der Blickrichtung(-en) wichtig für spätere Anwendung!

Anwendung einer Spherical Environment Map: - Zur Bestimmung der Texturkoordinaten eines dargestellten Punktes wird zuerst die Normale  $n$  an diesem Punkt bestimmt. - Die Normale  $n$  wird auf die  $x/y$ - Ebene projiziert. Die Koordinaten des projizierten Normalenvektors entsprechen den Texturkoordinaten in der Sphere Map, welche die an dieser Stelle reflektierte Umgebung zeigt. - Merke: Die Reflexion ist nicht von der Lage des reflektierenden Punktes abhängig (nur von der Normalenrichtung).

Environment Map in latitude-/longitude-Koordinaten: - Spiegelung wird aus Richtung des gespiegelten Strahls in Winkelkoordinaten (lat/long) berechnet - entweder pro Pixel (Pixel-Shader) oder pro Vertex mit anschließender (linearer) Interpolation pro Pixel - keine Berücksichtigung der Position des spiegelnden Objekts - korrekt nur für unendlich entfernte gespiegelte Objekte → geeignet zur Spiegelung weit entfernter Objekte (Landschaften, große Räume auf relativ kleinen Objekten)

High-dynamic Range Imaging (HDRI) Env-Maps: - enthalten "gesamte Dynamik" des Lichts (als Floating Point Farbwerte) - Wesentlich realistischere Bilder! - Tone Mapping: berechnete HDRI-Bilder werden anschließend auf die Dynamik des Monitors reduziert - Refraktion / Brechung mit Environment Maps: - wie Spiegelung, jedoch Sekundärstrahl aus Sehstrahl über Brechungsindex und Oberflächennormale, statt gespiegelt - Beispiel: Glas als Polygonflächen mit Rückseite + Normalen (2-fache Brechung!) + Spiegelung als Multi-Pass (Überlagerung zweier Effekte) - kann im Zusammenhang mit Cube-Maps, Spherical oder Lat/Long Environment Maps angewendet werden

Was? aus Originaltextur Bildung einer Menge jeweils kleinerer Texturen (halbe Kantenlänge)

Wozu? Vermeidung/Abmilderung von Aliasing-Effekten durch "Vorfilterung und Anwendung der passend aufgelösten Textur(-en) (1 Pixel 1 Texel) per bilinearer Filterung oder trilinearer Filterung

Aliasing-Effekte durch Koordinatentransformation: - Pixel der Textur und Pixel des dargestellten Bildes weisen (aufgrund der Bildtransformation) im Allgemeinen unterschiedliche Rastergrößen auf. - simpler Ansatz: Berechnung der transformierten Texturkoordinaten als Floating-Point-Werte und Rundung auf ganze Zahlen - bei inverser Transformation vom Zielbild zurück zur Textur dann keine Lücken im Bild, aber die Pixel der Textur können ausgelassen oder mehrfach verwendet werden (Bildpixel werden genau einmal angewendet) - durch das Resampling der Textur auf das resultierende Bildraster entstehen oft Aliasing-Artefakte Zwei wesentlich unterschiedliche Situationen: - Abbildung mehrerer Texturpixel auf ein Bildpixel (Unterabtastung) oder - Abbildung eines Texturpixels auf mehrere Bildpixel (Überabtastung) - Filteroperationen zur Interpolation der Bildpixel-Färbung in jedem Fall notwendig, insbesondere bei der Unterabtastung wird ein vorheriges Tiefpassfiltern und Resampling notwendig! - Ansonsten Verletzung des Abtasttheorems / Nyquistfrequenz!

Beispiel perspektivische Verkürzung der Schachbretttextur: - in Realität eigentlich starke Verkleinerung der Textur bei größerer Entfernung! - → Moiré Muster - Originaltextur ist an diesen entfernten Stellen im Bild zur Laufzeit nicht mehr erkennbar (Unterabtastung, aus mehreren Texeln, welche "hinter einem Pixel liegen", wird nur einer ausgewählt) - Treppenstufen im Nahbereich resultieren aus Überabtastung (mehrere Pixel teilen selben Texel) - Lösung: Textur muss vorher passend durch Tiefpassfilter in der Auflösung reduziert werden → Aufbau und Anwendung einer Mip-Map - Ziel der Mip-Map: stets 1 Texel pro Pixel bereitstellen - In 3D-Szenen können Körper mit der selben Textur vom Betrachter unterschiedlich weit weg sein. → im Bild oft Unterabtastung (Minification) oder Überabtastung (Magnification) und entsprechende Aliasing-Effekte durchs Resampling! - Ansatz: Vorberechnung derselben Textur für verschiedene Entfernungen - Stufe 1: volle Auflösung - Stufe 2: halbe Auflösung in jeder Richtung ( $1/2$ ) - ... - Stufe  $k$ : Auflösung  $(1/2)^k$  - Stufe  $n$ : niedrigste Auflösung (je 1 Pixel für z.B. R, G und B) - Speicherbedarf: - (hypothetische)

Annahme: Anordnung im Array (getrennt f. RGB) → Alle niedrigen Auflösungen verbrauchen zusammen nur ein Viertel des Speicherplatzes - Mip steht für lat. multum in parvo = viel (Information) auf wenig (Speicherplatz) - niedrige Auflösungsstufen werden durch Filterung aus den höheren berechnet: - einfach: z.B. Mittelwert aus 4 Pixeln (Box-Filter) oder - aufwendiger: z.B.: Gaußfilter (siehe Kap. Bildverarb.) - Beispiel: OpenGL-Filteroperationen im Bildraum (zur Laufzeit ausgeführt): - GL\_NEAREST: Annahme des Wertes des nächstliegenden Textur-Pixels - GL\_LINEAR: bilineare Interpolation: gewichteter linearer Durchschnitt aus einem  $2 \times 2$ -Feld der am nächsten liegenden Texturpixel - Genauere Interpolationsverfahren (z.B. bikubisch) gelten als zu aufwendig für Echtzeitanwendung - Beispiel für stark vergrößerte Textur: - Aus der Nähe betrachtet, wird das Texturraster auf dem Bildraster entsprechend skaliert (vergrößert). - durch Runden der Texturkoordinaten (d.h. ohne Filterung) - mit bilinearem Filter gewichtete Texturfarbwerte proportional zum Abstand vom gerundeten Koordinatenwert Aufbau der Mip-Map (als Vorverarbeitungsschritt beim Rendering): - Speicherung der Originaltextur - rekursive Speicherung der geringer aufgelösten Texturen (je  $1/2$  Kantenlänge) bis hinunter zu einem einzelnen Pixel Vorteile: - Filter-Operationen können bei Initialisierung der Textur vorausberechnet werden - nur ein Drittel zusätzlicher Speicherplatzbedarf

Darstellung mit Mip-Map Texturen (zur Laufzeit) - Auswahl der passenden Auflösungsstufe  $k$  skalierend berechnet aus der Entfernung zum Betrachter und der perspektivischen Verkürzung (siehe Kameratransf.):  
 $d/z = (1/2)^k \rightarrow k = \log_2(z) - \log_2(d)$  - Transformation der Pixel zwischen den Textur-Eckkoordinaten der gewählten Auflösung auf das Polygon im Bildraum - typ. Verwendung der linearen Filter zur Vermeidung von Aliasing-Effekten durch Trilineare Filterung: zusätzlich zu bilinearem Filtern in einer Mip-Map-Stufe wird linear gewichtet zwischen zwei Mip-Map-Stufen (auf-, bzw. abgerundete Werte von  $k$ ) interpoliert: z. B. wenn  $k = 2.3 \rightarrow 30\% \text{Anteil}_{k=3}$  und  $70\% \text{Anteil}_{k=2}$

Anti-Aliasing durch trilineare Filterung: - Durch die perspektivische Verkürzung wird eine weiter hinten liegende Textur verkleinert und im Vordergrund vergrößert. Bei einer Skalierung kleiner als 1 überspringt die gerundete inverse Texturtransformation Pixel in der Textur (minification). Die im Bildraum gesampelten Texturpixel werden somit "willkürlich ausgewählt. Dadurch können Treppenstufen und Moiré-Muster entstehen (Aliasing-Effekt: linkes Bild). Durch Mip-Mapping werden an diesen Stellen geringer aufgelöste (gefilterte) Texturen verwendet (Rechtes Bild: Mit Mip-Mapping und tri-linearer Filterung wird ein Anti-Aliasing-Effekt erreicht) - Vergrößerte Darstellung: Trilinearen Filterung = lineare Filterung zwischen den zwei aufeinander-folgenden (am besten passenden) Mip-Map-Stufen + bilineare Filterung in jeder der beiden Stufen. → Kantenglättung, Tiefpassfilter (Mittelwert / hier Grauwerte) Anisotrope Filterung: - z.B. bei flacher Aufsicht ist die Verkleinerung in  $y$ -Richtung viel stärker als in  $x$ -Richtung! -

Ohne spezielle Maßnahmen für diesen Fall müsste jeweils die Mip-Map-Stufe mit der kleinsten Auflösung verwendet werden, sonst treten wieder Aliasing-Artefakte auf! - → Dies führt zur unscharfen Texturabbildung. - Abhilfe: Anisotrope Mip-Maps (= Rip-Maps, Rectangular Mip-Maps)  
 Anisotropic Mip-Map (Rip-Map): - Verschiedene Auflösungsstufen in x- und y-Richtung werden erzeugt, sodass für jede Situation die richtige Auflösung gefunden werden kann ohne beim Resampling das Abtast-theorem zu verletzen. - Aber: Vierfacher Speicherbedarf gegenüber höchster Auflösung (statt 1,33 - s. MipMap)  
 - Reliefartige Texturen: Herkömmliche Texturen sehen aus der Distanz zwar akzeptabel aus, von Nahem betrachtet erscheinen sie flach. - Grund: keine korrekte 3D-Beleuchtung, Abschattung, keine Verdeckung, etc. - Idee: Verwendung zusätzlicher Texturen, welche Tiefeinformationen beinhalten  
 - Bump Map: Offset zur Polygonebene in Richtung der Normale als Grauwert“ der Textur kodiert - Polygon: als Schnitt mit Normalenrichtung - Anwendung des Offsets auf Polygonfläche (Drehung): Die Normale wird als Gradient der Bumpmap berechnet. Die Beleuchtung wird daraus wie bei der Normalmap pro Pixel berechnet. - Ein Offset“ wird nicht berücksichtigt! → Als Konturen nicht erkennbar!  
 - Normal-Map: Normalen Vektor x/y/z als RGB-Wert kodiert - Polygon: als Schnitt mit Normalenrichtung - Anwendung der Normal-Map auf Polygonfläche: Die Normale der N-Map modifiziert die Flächennormale (räumliche Drehung). Bei der Beleuchtungsberechnung wird für jedes Pixel die modifizierte Normale verwendet. - Ein „Offset“ wird nicht berücksichtigt! → Als Konturen nicht erkennbar!  
 - Parallax Map Tomomichi Kaneko et al. 2001 - Ausgangsdaten: Bump Map - Die u/v-Koordinaten der angezeigten Textur werden Entsprechend der Blickrichtung beim Look-up um  $\delta u = h * \tan(\phi)$  verschoben. Die daraus resultierende Verzerrung verstärkt den 3D-Effekt, allerdings

ohne korrekte Berücksichtigung der Verdeckung - Anwendung des Offsets auf Polygonfläche (Drehung): Anwendung der Bump Map des Offsets auf Polygonfläche (räuml. Drehung der Modellkoord.) Die Normale wird als Gradient der Bumpmap berechnet. Die Beleuchtung wird daraus wie bei der Normalmap pro Pixel berechnet.  
 - Ausgang: Wiederum Bump Map, jedoch Bestimmen des korrekten Schnitts eines Sehstrahls mit der Bump Map durch iterative Suche des Schnittpunktes - Finde  $u_0$ , sodass  $u - u' = h(u') * \tan(\phi)$  mittels Bisektion entlang dem Sehstrahl - Bei Mehrdeutigkeit: Finde  $u_0$  am weitesten weg von  $u$  → korrekte Verdeckung - Silhouetten: Auch u/v-Koordinaten außerhalb der Polygongrenzen müssen berücksichtigt werden! - aufwendige Shader Programme nötig  
 - DECAL (Abziehbild) RGBA-Werte ohne Berücksichtigung der Beleuchtung (emmisiv, evtl. mit Alpha Wert (A) für transparente Anteile) - DIFFUSE: RGB-Werte werden als diffuser Farbanteil mit Beleuchtung verrechnet - Graustufen: Helligkeitswert wird mit dem diffusen Materialfarben multipliziert. - Specular Map: Wie bei Diffuse Texture Map, jedoch für spekulären Anteil - Normal Map: Normalisierte Normalenrichtung (als 2farbiges Rasterbild). Dient zur Modulierung der Flächennormalen und wird bei der Beleuchtung berücksichtigt. Farbwerte kommen aus der Materialkonstante des Polygons, oder aus der Diffuse Map (bzw. Specular Map). Ergibt aus der Ferne eine dreidimensionalen (reliefartige) Struktur. - Bump Map: Statt der Normalen wird eine Erhöhung (in Richtung der Normalen) kodiert (grauwertiges Rasterbild). Die Normalenrichtung wird daraus als Gradient (Differenz zweier benachbarter Pixel) bei der Darstellung abgeleitet. Danach Beleuchtung wie Normal Map. - Parallax Map: zusätzlich Pixelverschiebung als Funktion der Höhe und Kamerarichtung  
 1. Durchgang: - Erzeugen der Shadow Map - Darstellung (mit z-Werten) aus Sicht der Lichtquelle - Kamera Koordinaten in

der Lichtquelle zentriert (Matrix L) - z-Puffer als Textur speichern 2. Durchgang: - Kamera Ansicht: View Matrix: V (ebenfalls mit z-Puffer) - → Um den Schatten zu erzeugen benötigen wir Shader mit Lookup in der Shadow Map-Textur: - 4x4-Matrix:  $M = V^{-1} * L$   
 Shadow map look-up: - Transformiere jedes Pixel aus dem Kameraraum in den Lichtraum -  $p' = L * V^{-1} * p$  - Vergleiche transformierte z-Werte ( $p'_z$ ) mit den z-Werten der Shadow Map ( $z_s$ ) - ( $p'_z > z_s$ ): im Schatten - keine Beleuchtung von der Lichtquelle - sonst: Punkt ist von der Lichtquelle her sichtbar, wende Beleuchtung in der Schattierung des Pixels an  
 Z-fighting beim Schattentest: - Schattentest ( $p'_z <= z_s$ ) sollte für beleuchtete Pixel korrekt ( $p'_z = z_s$ ) ergeben. - Aufgrund der Rechenungenauigkeit der Fließkomma-Arithmetik wird Gleichheit selten erreicht! - Beleuchtete Polygone schatten sich teilweise selbst ab. - Lösung: kleiner Offset im Schattentest:  $IF(p'_z <= z_s + Offset..)$  - durch das Offset wird sichergestellt, dass keine falschen Schatten entstehen  
 Uniform Shadow-Map - Probleme: zu niedrige Auflösung der Shadow Map im Nahbereich, Großteil der Shadow Map ist irrelevant für Kameraansicht  
 Perspektive Shadow-Map - adaptive schiefsymmetrische Projektion; nicht uniforme perspektive Shadow Map  
 - Transformation des Texturraums in den Bildraum der Darstellung: - Verwendung unterschiedlicher geometrische Transformationen (z. B. affin, perspektivisch, Env. Maps, etc.)  
 - Anwendung immer als inverse Transformation! - Resampling + Rekonstruktion: Das transformierte Texturraster wird nach der Transformation durch das Bildraster neu abgetastet. - Filter: Verhindern bzw. Abmildern von Aliasing-Effekten, verursacht durch Resampling. - Lösung: Tiefpass-Filter vor der Transformation: Mipmapping, Anisotrope Filter. - Beim Abtasten (Rekonstruktion): Trilineare Filterung in x, y, und k (Mip-Map-Stufe) - Texturinhalt als Material, Beleuchtung, Geometrie interpretiert